



Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote, sistema operativo Openwsn y TheThings.io

Rubén Cabello Chacón

Master Universitario de Ingeniería de Telecomunicación UOC-URL

José López Vicario

Xabi Vilajosana Guillen

07/01/2018



Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-SinObraDerivada
[3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)
Reconocimiento-CompartirIgual [3.0 España de](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)
[Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote, sistema operativo Openwsn y TheThings.io</i>
Nombre del autor:	<i>Rubén Cabello Chacón</i>
Nombre del consultor/a:	<i>Jose Lopez Vicario</i>
Nombre del PRA:	<i>Xavier Vilajosana Guillen</i>
Fecha de entrega (mm/aaaa):	01/2018
Titulación:	<i>Master Universitario en Ingeniería de Telecomunicación (UOC-URL)</i>
Área del Trabajo Final:	<i>Telemática</i>
Idioma del trabajo:	<i>Castellano</i>
Palabras clave	<i>IoT Openwsn OpenMote</i>
<p>Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i></p>	
<p>Los sistemas IoT requieren de un procesamiento de datos ya sea para poder realizar un control sobre los datos adquiridos o una actuación sobre otro elemento IoT basándonos en la información recopilada por nuestros sensores.</p> <p>Para este tipo de tratamiento existen distintas soluciones, en este trabajo se realiza un estudio del tratamiento y procesado bajo cloud computing donde se realiza el envío y tratamiento a un servicio cloud donde se realizan las tomas de decisiones y otro procesamiento tipo fog todo se realiza en un elemento edge dentro de la propia red IoT donde se realizan todas las acciones sobre la información.</p> <p>En el trabajo se desarrolla un sistema bajo OpenWSN donde una serie de scripts bajo Python realizan la generación de datos aleatorios de distintos sensores de la placa OpenMote, los cuáles serán almacenados en una base de datos y tratados en una Raspberry Pi en el modelo Fog computing y en la nube utilizando Cloud computing.</p> <p>Se presenta un estudio y comparativa de ambos donde se muestra las ventajas y desventajas de cada una en términos de retardos, coste y consumo de ancho de banda, llegando a la conclusión de que sistema es más adecuado para aplicaciones IoT en tiempo real.</p>	

Abstract (in English, 250 words or less):

IoT Systems require a data processing system for having a control over the gathered information or an actuation over another IoT element based upon the data compiled from our sensor network.

For this kind of treatment there are several solutions, in this paper the study is based on the next: cloud computing where the data treatment y process is done completely into the cloud where decisions are made. The other is fog computing all is done into an edge element inside the own IoT network where all actions over the information are performed.

During this job a system under OpenWSN is developed where resides several Python scripts which perform the random data generation simulation them from different OpenMote devices, the data will be stored into a database and treated in a Raspberry Pi using the fog computing model and into the cloud using Cloud computing.

A final study is showed finally and a comparative between both systems where the advantages and disadvantages of both in terms of delay, cost and bandwidth usage are shown, reaching the conclusion of which computing system is more suitable for IoT applications in real time.

Índice

1. Introducción.....	1
1.1 Contexto y justificación del Trabajo	1
1.2 Objetivos del Trabajo	1
1.3 Enfoque y método seguido	2
1.4 Planificación del Trabajo.....	3
1.5 Breve resumen de productos obtenidos.....	4
1.6 Breve descripción de los otros capítulos de la memoria	5
2. Internet of Things (IoT).....	5
2.1 Introducción.....	5
2.1.1 Arquitectura IoT.....	6
2.1.2 Protocolos IoT.....	8
2.1.3 Modelos de conectividad en IoT.....	11
2.1.4 Campos de aplicación IoT.....	14
2.2 IEEE 802.15.4.....	15
2.2.1 Red IEEE 802.5.14.....	18
2.3 IEEE 802.15.4e.....	22
2.4 CoAP.....	24
2.3 Fog computing.....	27
2.6 Cloud computing.....	29
2.7 Comparación Cloud computing – Fog computing.....	31
2.8 Sistemas operativos para IoT.....	33
2.8.1 OpenWSN.....	34
2.8.2 OpenSim.....	34
2.8.3 OpenVisualizer.....	35
2.8.4 Librería Python CoAP.....	36
2.9 Plataformas en la nube.....	37
3. Diseño e implementación del sistema.....	38
3.1 Arquitectura del sistema.....	39
3.2 Configuración de software.....	40
3.2.1 Instalación OpenWSN.....	40
3.2.2 Simulación en OpenSim.....	42
3.3 Hardware.....	47
3.4 Compilación sobre OpenMote.....	48
3.5 Instalación BBDD en Raspberry Pi.....	52
3.6 Descripción del firmware.....	52
3.6.1 Scripts Python.....	52
3.6.2 Base de datos.....	60
3.7 Subida de datos a la nube.....	62
4. Resultados comparativa Fog – Cloud Computing.....	67
5. Conclusiones.....	69
6. Glosario.....	71
7. Bibliografía.....	73

Índice de figuras

Figura 1. Diagrama Gantt.	3
Figura 2. Fechas realización TFM.	4
Figura 3. IoT desde el punto de vista humano[3].	6
Figura 4. Arquitectura IoT[4].	7
Figura 5. Pila de protocolos IoT[6].	8
Figura 6. Comparación protocolos comunicación[6].	11
Figura 7. Comunicación equipo a equipo[8].	12
Figura 8. Comunicación equipo a Cloud[8].	13
Figura 9. Comunicación equipo a Gateway[8].	13
Figura 10. Compartición de datos con Back-End[8].	14
Figura 11. Topologías IEEE 802.15.4[13].	15
Figura 12. PHY IEEE 802.15.4[13].	16
Figura 13. IEEE 802.15.4 Data Link Layer[13].	16
Figura 14. Supertrama[13].	17
Figura 15. Network Setup[13].	18
Figura 16. Coordinador a nodo[13].	20
Figura 17. Mecanismo de interface MAC[13].	21
Figura 18. TSCH timeslot[14].	23
Figura 19. IEEE 802.15.4e RIT[14].	24
Figura 20. IEEE 802.15.4e CSL[14].	24
Figura 21. Mensaje CON CoAP[15].	25
Figura 22. Mensaje NON CoAP[15].	25
Figura 23. Formato mensaje CoAP[16].	26
Figura 24. Dispositivos en Fog computing[18].	28
Figura 25. Arquitectura Fog computing[20].	29
Figura 26. Servicios Cloud Computing.	30
Figura 27. Stack protocolos OpenWSN[22].	34
Figura 28. Red OpenWSN simulada[27].	35
Figura 29. OVS vía web[23].	36
Figura 30. Estados Python CoAP[28].	37
Figura 31. Arquitectura del sistema[29].	40
Figura 32. Uso de Copper a un mote simulado.	42
Figura 33. Compilación simulación mote.	42
Figura 34. Ejecución OVS.	43
Figura 35. Motes de una simulación.	44
Figura 36. DAGRoot.	44
Figura 37. Vecinos DAGRoot.	44
Figura 38. Topología.	45
Figura 39. Routing.	45
Figura 40. RPL DAO.	46
Figura 41. Icmp IPv6.	46
Figura 42. Raspberry Pi 3[33].	47
Figura 43. OpenMote[34].	47
Figura 44. OpenUSB con OpenMote[35].	48
Figura 45. Opendesf.h.	49
Figura 46. Sconscript.env.	49
Figura 47. Sconscript.	50

Figura 48. Openapps.c	50
Figura 49. Compilación OpenMote.	51
Figura 50. Captura de datos mediante Copper.....	51
Figura 51. Versión MariaDB.	52
Figura 52. Esquema Scripts Python.	52
Figura 53. Start.py.....	53
Figura 54. dBclass.py.....	54
Figura 55. RandomIoT.py.....	55
Figura 56. Timer.py.	56
Figura 57. CConnection.py.	57
Figura 58. ActionTrigger.py.	57
Figura 59. nohup.	58
Figura 60. Ejecución Scripts.....	59
Figura 61. IoTDB.....	60
Figura 62. Agregación de datos.	61
Figura 63. Creación del thing.....	62
Figura 64. Código envío datos con token.	63
Figura 65. Creación Dashboard.	64
Figura 66. Dashboards distintos sensores.	65
Figura 67. Job y Trigger.....	65
Figura 68. Job.....	66
Figura 69. Trigger.	67
Figura 70. Resultado Trigger con Twilio.....	67
Figura 71. Latencia a Thethings.io	68

Índice de tablas

Tabla 1. Protocolos comunicación.....	10
Tabla 2. Rangos frecuenciales.	15
Tabla 3. CoAP Status Code.	27
Tabla 4. Modelos nube Cloud computing.....	31
Tabla 5. Qos Cloud computing vs Fog computing.	32
Tabla 6. Delay Cloud computing vs Fog computing.	32
Tabla 7. Características librería CoAP.....	36

1. Introducción

1.1 Contexto y justificación del Trabajo

En la actualidad el concepto de IoT (Internet of Things) se está popularizando y convirtiéndose en una de las tendencias más importantes en la industria durante los últimos años. Este tipo de soluciones basadas en IoT han tenido un gran auge gracias a que la capacidad de conectividad y computación se ha vuelto más eficiente en términos energéticos, así como la gran miniaturización que han sufrido, proporcionan el aumento y la demanda de dichos sistemas.

Hace unos años mediante plataformas como Arduino o Raspberry pi se podía conectar distintos tipos de sensores para realizar monitorizaciones de temperatura, humedad, etc. siempre abierto a la imaginación del usuario final, ahora debido a la gran cantidad de datos a procesar y analizar se incluye las soluciones basadas en la nube para IoT aportando flexibilidad, analíticas en tiempo real, estado de nuestra red sensorial, acceso desde terminales móviles, etc.

Existen plataformas propietarias que realizan las funciones descritas anteriormente, en este proyecto se utilizará un sistema basado en código abierto para permitir la edición y modificación del mismo dentro de todas las posibilidades que surjan y poder adaptarlo a nuestras necesidades, no como en un sistema de código cerrado por otra parte se utilizará la plataforma Thethings.io donde se almacenarán los datos en la nube para su tratamiento, así como permitir la visualización del sistema desde cualquier punto con conexión a Internet.

Una vez obtenidos los datos de los sensores mediante la generación de datos aleatorios bajo scripts en Python se estudiará dos tipos distintos de procesado de datos, uno bajo cloud computing, donde los datos son tratados en la nube y se realizarán las tomas de decisiones y otra basada en fog computing, todo el procesado y toma de decisiones se realiza en un equipo edge antes de enviar la información a la nube.

Con estas dos soluciones se realizará un estudio de ambas soluciones donde se determinará qué solución se amolda mejor a un entorno IoT.

1.2 Objetivos del Trabajo

El objetivo del trabajo fin de Master es el desarrollo e implementación de un prototipo demostrador de IoT utilizando el estándar IEEE 802.15.4e, bajo el uso de hardware OpenMote y Raspberry pi, el sistema operativo Openwsn y la plataforma Thethings.io.

La función de este prototipo es la adquisición de distintos datos de los sensores incluidos en el hardware del mote mediante el protocolo CoAP, estos datos serán procesados y almacenados en una base de datos en una Raspberry pi y luego mediante un API enviados al cloud de Thethings.io donde los datos serán tratados para su visualización.

Se buscan los siguientes objetivos individuales para obtener el resultado final:

- Documentación sobre IoT, comprensión del concepto, características, arquitectura y campos de aplicación actuales.
- Estudio de OpenMote, sus componentes, características y funcionalidades.
- Documentación sobre el estándar IEEE 802.15.4 e IEEE 802.15.4e, para poder adquirir un conocimiento sobre los mismos, así como las mejoras introducidas en IEEE 802.15.4e.
- Diseño de un prototipo que cumpla con las características de IoT.
- Implementación de software bajo Openwsn, así como documentación del propio sistema.
- Desarrollo de software para poder adquirir, almacenar y tratar los datos en Thethings.io
- Creación de aplicación en Phyton para ejecutar en una Raspberry pi que será el punto central de la arquitectura.
- Estudio y comparación de cloud computing y fog computing.

1.3 Enfoque y método seguido

La planificación del trabajo se basa en las PEC planificadas, mediante las entregas programadas se planifican una serie de tareas a tener completadas para cada entrega durante la duración del semestre, siguiente un diagrama Gantt a desarrollar durante el desarrollo del trabajo.

Se ha utilizado código OpenMote en C disponible para su utilización en github de OpenWSN para poder obtener la información de los sensores siendo posible su visualización mediante el pluggin Copper de Firefox, al haberme encontrado con distintos problemas a la hora de realizar la lectura mediante Python la parte de tratamiento y procesado de los datos se realiza bajo generación de datos aleatorios generados por script bajo Python.

Una vez obtenidos los datos aleatorios de temperatura y humedad se realizan dos tipos distintos de procesado para su estudio y comparación, uno bajo cloud computing y otro utilizando fog computing.

1.4 Planificación del Trabajo

Se describe en el siguiente diagrama de Gantt los pasos e hitos para la realización del TFM.

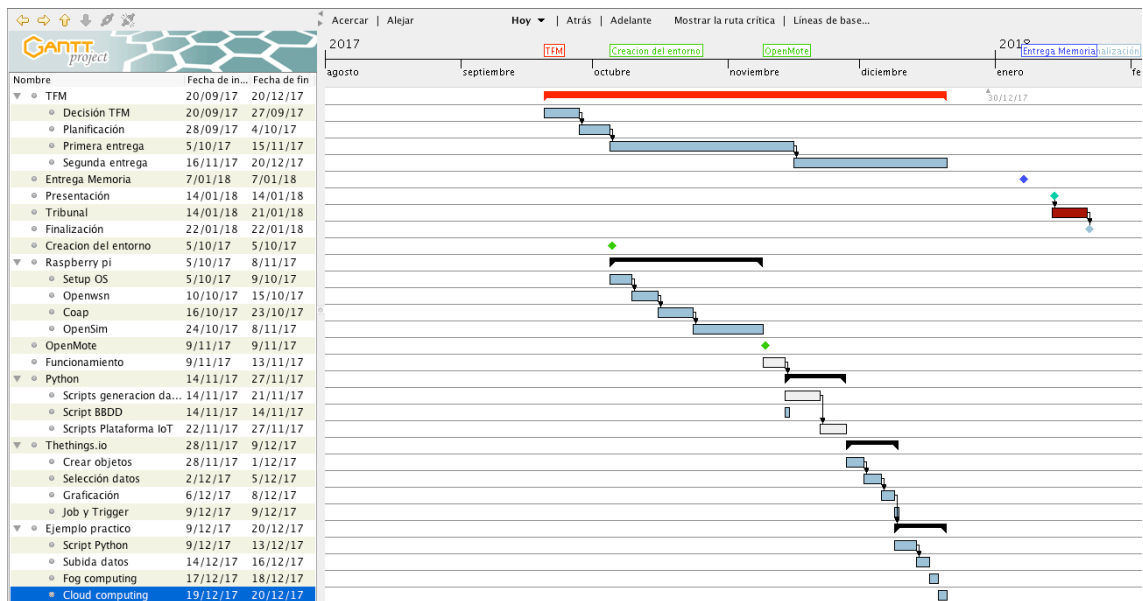


Figura 1. Diagrama Gantt.

El diagrama Gantt ha ido variando durante la creación del proyecto debido básicamente a problemas encontrados a la hora de obtener una lectura mediante scripts en Python de los mote, el problema se encuentra en un error en las librerías de Python al utilizar CoAP para realizar la lectura de los mote.

La modificación que se hace es utilizar scripts bajo Python que realicen una generación aleatoria de información como si esta fuera obtenida directamente de los sensores para su posterior tratamiento ya sea bajo fog computing o cloud computing.

Nombre	Fecha de inicio	Fecha de fin	Recursos
TFM	20/09/17	20/12/17	
Decisión TFM	20/09/17	27/09/17	
Planificación	28/09/17	4/10/17	
Primera entrega	5/10/17	15/11/17	
Segunda entrega	16/11/17	20/12/17	
Entrega Memoria	7/01/18	7/01/18	
Presentación	14/01/18	14/01/18	
Tribunal	14/01/18	21/01/18	
Finalización	22/01/18	22/01/18	
Creacion del entorno	5/10/17	5/10/17	
Raspberry pi	5/10/17	8/11/17	
Setup OS	5/10/17	9/10/17	
Openwsn	10/10/17	15/10/17	
Coap	16/10/17	23/10/17	
OpenSim	24/10/17	8/11/17	
OpenMote	9/11/17	9/11/17	
Funcionamiento	9/11/17	13/11/17	
Python	14/11/17	27/11/17	
Scripts generacion datos	14/11/17	21/11/17	
Script BBDD	14/11/17	14/11/17	
Scripts Plataforma IoT	22/11/17	27/11/17	
Thethings.io	28/11/17	9/12/17	
Crear objetos	28/11/17	1/12/17	
Selección datos	2/12/17	5/12/17	
Graficación	6/12/17	8/12/17	
Job y Trigger	9/12/17	9/12/17	
Ejemplo practico	9/12/17	20/12/17	
Script Python	9/12/17	13/12/17	
Subida datos	14/12/17	16/12/17	
Fog computing	17/12/17	18/12/17	
Cloud computing	19/12/17	20/12/17	

Figura 2. Fechas realización TFM.

1.5 Breve sumario de productos obtenidos

A continuación, se detallan los productos obtenidos durante la realización de este TFM:

- Sistema operativo IoT con funcionalidad de simulación.

- Scripts Python para generación de datos aleatorios de temperatura y humedad.
- Aplicación en Python para realizar un grabado de datos en una base MySQL.
- Script para comunicación con Thethings.io.
- Visualización en Thethings.io de los datos generados.
- Estudio y comparativa de cloud computing y fog computing.

1.6 Breve descripción de los otros capítulos de la memoria

La memoria se estructura en los siguientes capítulos:

- Capítulo 2: se proporciona una introducción de los distintos componentes tecnológicos abarcados en el TFM tales como IoT, fog computing, servicios en la nube e IEEE 802.15.4e.
- Capítulo 3: en este capítulo se realiza una descripción del sistema desarrollado, tanto de su componente de hardware como software, desde la configuración inicial del sistema, compilación, simulación y funcionamiento del prototipo.
- Capítulo 4: análisis de las pruebas del prototipo, fog computing vs cloud computing.
- Capítulo 5: Conclusiones.
- Capítulo 6: Glosario.
- Capítulo 7: Bibliografía.

2. Internet of Things (IoT).

2.1 Introducción.

Podemos definir IoT como una red de equipos físicos y otros tipos embebidos con electrónica, software, sensores, etc. con conectividad de red para poder intercambiar y recopilar información.

El término de IoT no se acuñó oficialmente hasta 1999 por Kevin Ashton[1] en relación a que la identificación RFID es un prerequisite para el Internet de las cosas.

Aunque anteriormente ya surgieron ejemplos de conexión de objetos cotidianos a la red como en 1980[2] en la universidad de Carnegie Melon unos programadores locales conectaron a Internet una máquina de Coca Cola para poder saber si la máquina tenía bebidas disponibles y si estaban frías.

Ahora mismo el término se utiliza como una conexión de dispositivos, servicios, sistemas y aplicativos con un gran campo de aplicación, no solamente la tradicional comunicación máquina a máquina (M2M).

Abarca una amplia gama de equipos y tecnologías diferentes por lo que no se puede enfocar en una tecnología o aspecto en concreto sino en una rica variedad de aplicaciones siempre abierta a la imaginación del usuario, se podría decir que la definición es bastante abstracta no pudiendo centrarse en un significado concreto.

En el IoT las cosas están interconectadas entre sí realizando tareas de adquisición, envío y tratamiento de información a partir de la cual se obtienen diversos resultados, tratamiento de información con fines estadísticos, ejecución de actuadores dependiendo de diferentes condiciones, etc.

2.1.1 Arquitectura IoT.

Uno de los retos cuando se planea un proyecto de IoT es el tratamiento de la complejidad, una solución típica IoT incluye muchos equipos heterogéneos con sensores que recopilan información para luego tratarla. Los equipos IoT pueden conectarse directamente a una red o utilizar un Gateway para dicha conexión, lo que proporciona la funcionalidad de comunicarse con servicios cloud y aplicaciones.

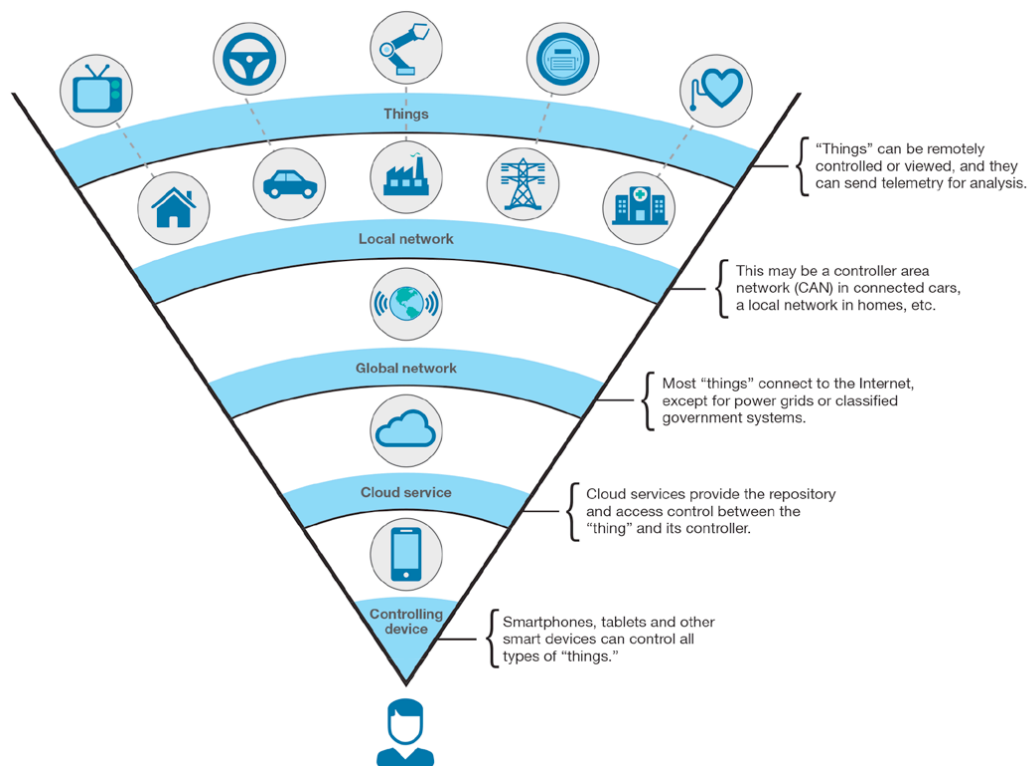


Figura 3. IoT desde el punto de vista humano[3].

Una arquitectura describe la estructura de la solución IoT, incluyendo los aspectos físicos (sensores, actuadores, Gateways, etc.) como los aspectos virtuales (protocolos de comunicación, servicios, etc.).

En la siguiente figura se muestra una arquitectura IoT.

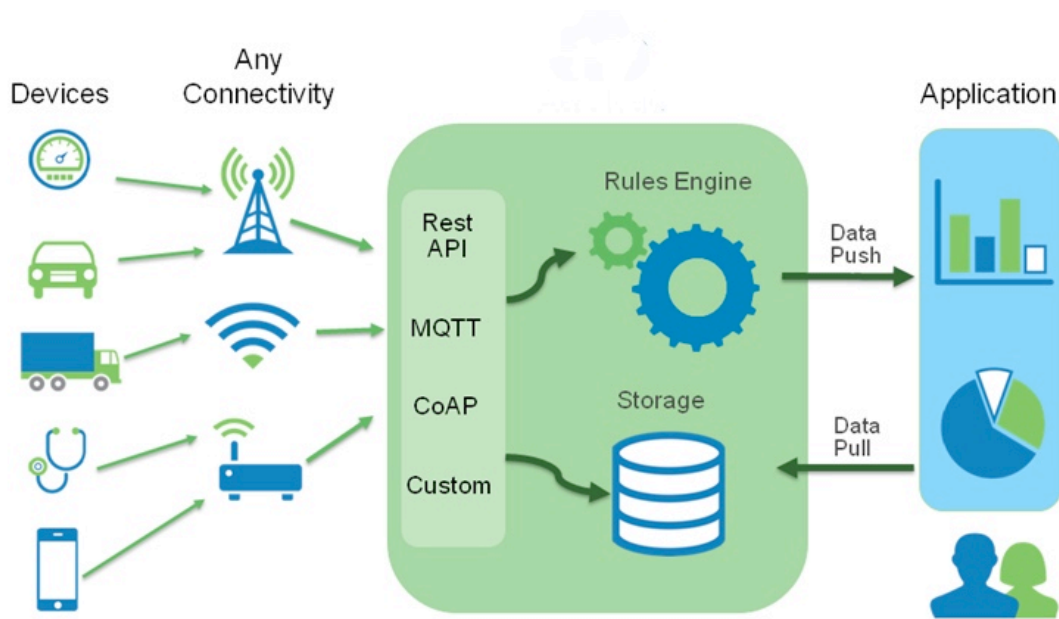


Figura 4. Arquitectura IoT[4].

Podemos definir las siguientes capas[5]:

- **Capa de dispositivos:** se trata de la primera capa en la arquitectura IoT, donde están localizados los sensores y actuadores físicos que están conectados a equipos IoT. Estos sensores y actuadores no se consideran inteligentes ellos mismos sino que tienen conexión vía ZigBee o Bluetooth LE a equipos con mayor capacidad de procesamiento.

La funcionalidad de los dispositivos en esta capa es la de realizar medidas de una magnitud física (temperatura, humedad, nivel lumínico, etc.), y enviar esta información a dispositivos de mayor inteligencia para ser procesada.

- **Capa de red:** encargada de la transmisión, recepción, enrutamiento y control de los datos transmitidos entre los dispositivos y nuestro punto central de la red. La conexión ya sea cableada o inalámbrica ha de cumplir con los estándares de calidad en términos de robustez, alto rendimiento. Al poder ser redes de tipo privado, públicas o híbridas han de cumplir una serie de requisitos a nivel de latencia, jitter, ancho de banda, seguridad y consumo energético.
- **Capa Edge:** relacionada a la analítica y pre-procesamiento de los servicios localizados en el borde de la red, estas analíticas son en tiempo real mediante el procesamiento del flujo de datos en el punto donde se recopila la información proporcionada por los sensores.

Se utilizan diferentes herramientas para poder extraer la información que nos interesa del flujo de datos transmitido por los dispositivos ya sea almacenando los datos en memoria RAM, en disco físico o vía streaming analizando en tiempo real.

- Capa de aplicación: donde la información capturada por nuestros dispositivos y ya tratada y transformada en nuestra capa Edge se analiza o se utiliza por determinadas aplicaciones dependiendo de la funcionalidad que se quiera dotar al sistema conjunto.

2.1.2 Protocolos IoT.

Podemos realizar una segmentación de los protocolos utilizados en IoT como un modelo de arquitectura como OSI[6], se definen las siguientes capas:

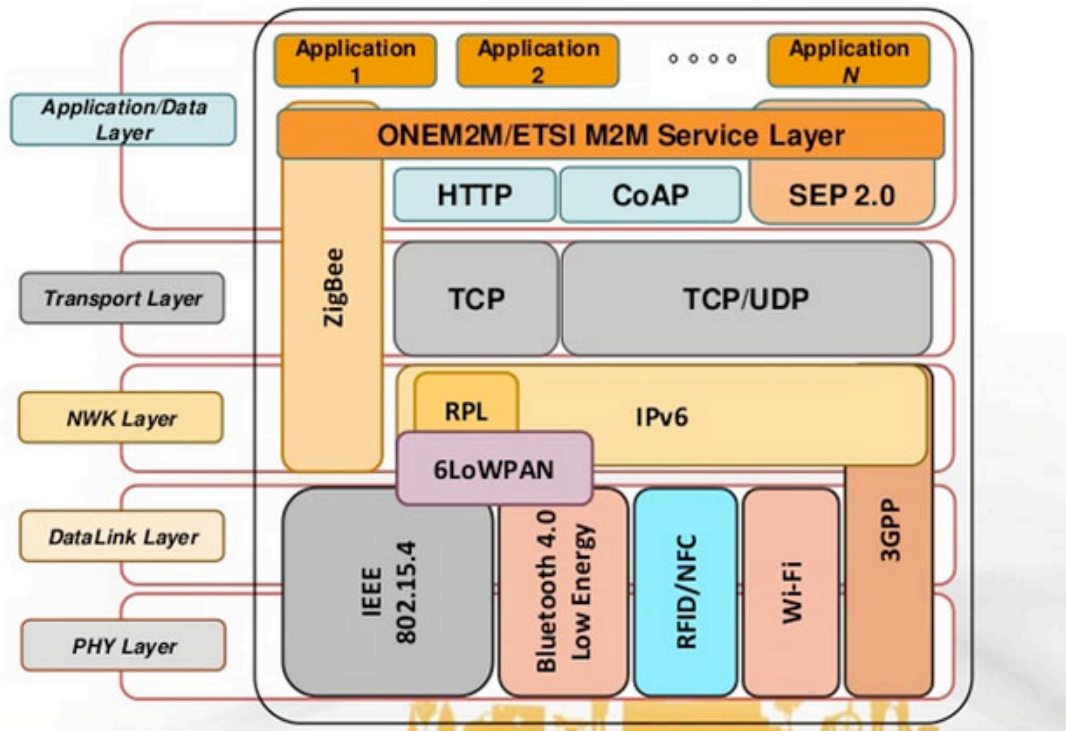


Figura 5. Pila de protocolos IoT[6].

- Infraestructura:
 - IPv6: protocolo de internet provee transmisión de datagramas end-to-end.
 - 6LoWPAN: IPv6 sobre redes personales inalámbricas de bajo consumo, opera en 2,4GHz y transferencia de 250 kbps.
 - UDP (User Datagram Protocol): un protocolo simple del modelo OSI en la capa de transporte para aplicaciones de red cliente/servidor, utilizado para aplicaciones en tiempo real.
 - uIP: un stack open source de TCP/IP para ser utilizado con microcontroladores de 8 y 16 bit.
 - DTLS (Datagram Transport Layer): provee privacidad en la comunicación para protocolos de datagrama, permite a aplicaciones cliente/servidor comunicarse de manera segura.

- TSMP (Time Synchronized Mesh Protocol): protocolo de comunicaciones para redes auto-organizadas o equipos inalámbricos denominados motes. TSMP permanece sincronizado a cada uno y se comunica mediante time slots, parecido a sistemas TDM.

- Descubrimiento:
 - mDNS (multicast Domain Name System): resuelve nombre de host a direcciones IP en redes pequeñas que no tienen un servidor de resolución de nombres.
 - HyperCat: un catálogo hipermedia basado en JSON para exponer una colección de URIs.
 - UPnP (Universal Plug and Play): una colección de protocolos de red que permite a dispositivos descubrir sin problemas la presencia de otros en la red y establecer funciones de compartición de datos, comunicación, etc.

- Protocolos de datos:
 - WebSocket: desarrollado como parte de la iniciativa HTML5, introduce el WebSocket de la interfaz JavaScript, define un único socket de conexión bidireccional sobre el que se pueden enviar mensajes entre cliente/servidor.
 - SSI (Simple Sensor Interface): un sencillo protocolo de comunicación diseñado para la transferencia de datos entre ordenadores o terminales de usuario con sensores inteligentes.
 - LWM2M[5]: es un sistema estándar en Open Mobile Alliance. Incluye DTLS, CoAP, Block Observe, SenML y recursos de directorio y los utiliza en una interfaz equipo-servidor.
 - LLAP (Lightweight Local Automation Protocol): es un mensaje simple y corto que se envía entre objetos inteligentes usando texto normal, puede funcionar sobre cualquier medio de comunicación.
 - DDS (Data-Distribution Service for Real-Time Systems): el primer middleware internacional de open source directamente publica-suscribe comunicaciones en tiempo real y sistemas embebidos.
 - XMPP (Extensible Messaging and Presence Protocol): tecnología abierta para comunicaciones en tiempo real, que refuerza aplicaciones como mensajería instantánea, voz y video llamadas.
 - MQTT (Message Queuing Telemetry Transport): este protocolo habilita un modelo de mensajes de publicación/suscripción en un modo extremadamente ligero. Es muy útil para conexiones en localizaciones remotas donde el ancho de banda es muy escaso.

- CoAP[6] (Constrained Application Protocol): protocolo en la capa de aplicación para su uso en dispositivos de internet con recursos restringidos. Está diseñado para traducir fácilmente HTTP para una integración simplificada en la web mientras cumple otros requerimientos como multicast y simplicidad. Utiliza métodos GET, PUSH, POST y DELETE, también utiliza parámetros DTLS para seguridad.
- Protocolos de comunicación:
 - IEEE 802.15.4e: estándar bajo el que se define el nivel físico y control de acceso al medio de redes inalámbricas de área personal (PAN), donde existen bajas tasas de transmisión de datos. Es la base para sistemas ZigBee, ISA100.11^a y MiWi.
 - ZigBee: utiliza el estándar 802.15.4, opera a 2,4GHz y ancho de banda de 250kbps, soporta un número máximo de nodos de 1024 con un rango de 200 metros, puede utilizar encriptación AES128.
 - LoRaWan: protocolo de red para redes Wireless donde los sensores son operados por baterías en una red regional, nacional o global.
 - Celular: tecnologías de comunicación inalámbricas ya sea GPRS, 2G, 3G o 4G.

En la siguiente tabla se muestra una comparativa de distintos protocolos de comunicación:

Tecnología	LTE	LoRaWAN	SigFox	WiFi	ZigBee
Frecuencia	LTE Frec.	Sub-bandas Reg.	Sub-bandas Reg.	2,4 - 5,8Ghz.	2,4Ghz.
DL	20 MHz.	125Khz - 500 Khz.	100Hz channel	20, 21, 22, 23, 24, 40, 80, 160 MHz	600 kHz, 1.2 MHz
UL			200Hz channel		
Acceso UL	OFDMA	CSS	UNB/FHSS	OFDM, DSSS, OFDMA	CSMA/CA
Acceso DL	SC-FDMA		UNB/FHSS		
Modulación DL	QPSK, 16QAM	LoRa, (G) FSK	GFSK	CCK, BPSK, QPSK, 16-QAM, 64-QAM	DSSS, BPSK, O-QPSK
Modulación UL			DBPSK		
Pico de datos	1 Mbps	0,3Kbps - 50Kbps	100bps/600bps	11-54-600Mbps	20 kbps, 40 kbps
Máx. cobertura	141 dB	150-157dB	146-162dB	200 m.	100 m.

Tabla 1. Protocolos comunicación.

Existe una extensa variedad de protocolos que serán aplicables dependiendo del tipo de sistema a desarrollar, en nuestro trabajo utilizaremos Ipv6 debido a que es el estándar de direccionamiento para IoT asignado una IP única a cada dispositivo sin necesidad de realizar NAT por problemas de duplicidad de direccionamiento, UDP para el transporte de información entre los sensores y el colector de información, menos retardo al no enviar paquetes ACK, CoAP por ser un protocolo de fácil utilización y poco peso en la comunicación y nos comunicaremos con IEEE 802.15.4e para el acceso al medio mediante una baja transmisión de datos y consumo.

Cada uno de los distintos protocolos de comunicación tienen distintas características de ancho de banda, distancia efectiva y consumo de energía como se puede observar en la siguiente figura:

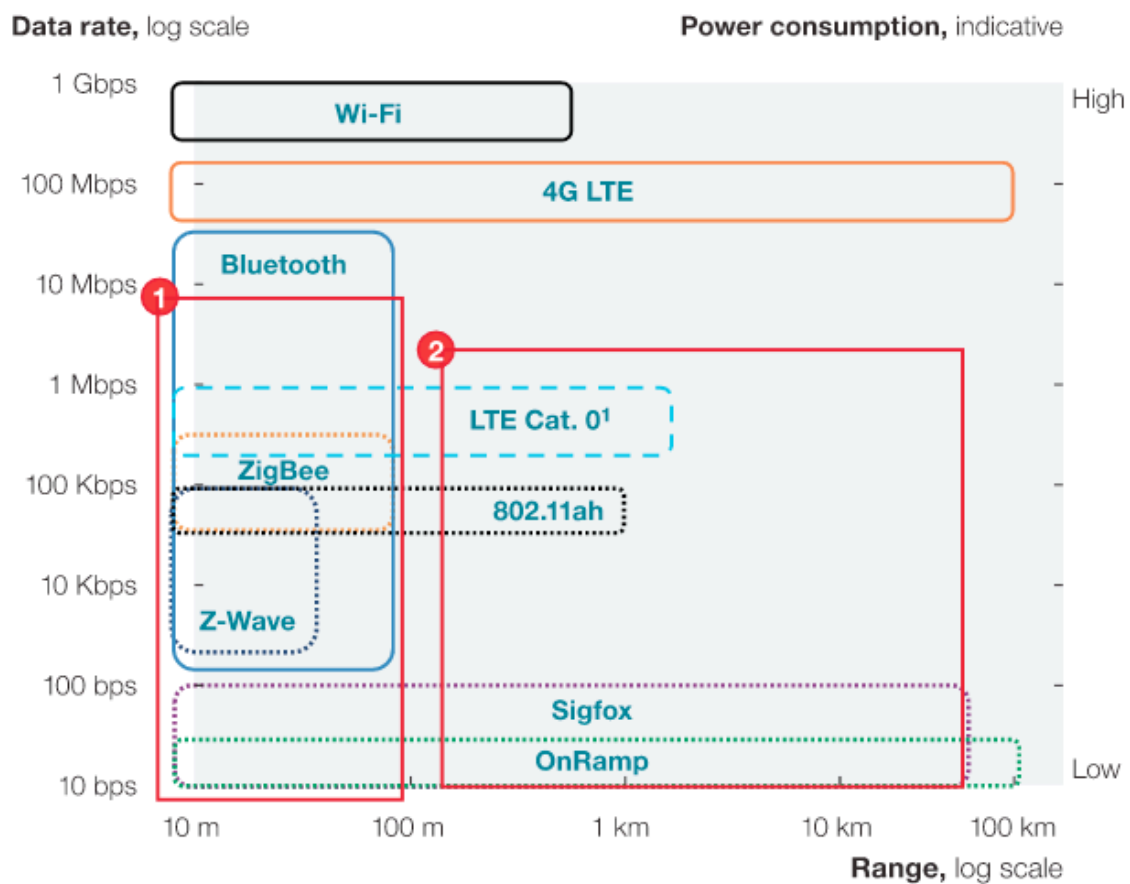


Figura 6. Comparación protocolos comunicación[6].

2.1.3 Modelos de conectividad en IoT.

Desde la perspectiva más básica IoT se trata de poder realizar la conexión de distintos equipos y sensores a Internet, aunque no siempre es obvio como realizarlo, por ello en Marzo de 2015 el equipo de Internet Architecture Board[7] creó una guía donde detallan los distintos modelos de conexión en IoT[8].

- Equipo a equipo: representa dos o más equipos que se conectan directamente y se comunican entre ellos. Pueden realizar la comunicación sobre distintos tipos de redes, como IP, ZigBee o Z-Wave.

Este modelo se utiliza comúnmente para sistemas de automatización doméstica donde se transfieren paquetes de información pequeños y se requiere poco ancho de banda, como control de luces, puertas, etc.

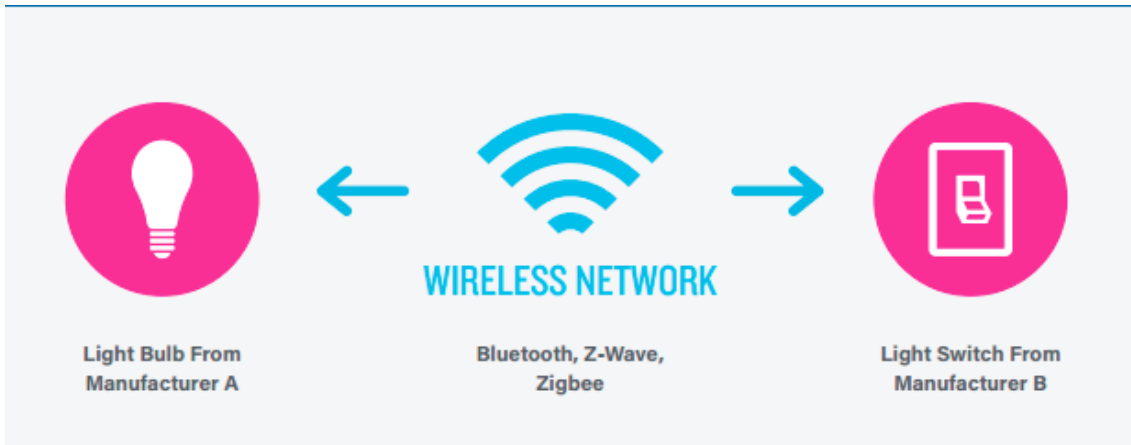


Figura 7. Comunicación equipo a equipo[8].

- Equipo a Cloud: esta clase de comunicación involucra un elemento IoT conectado directamente a Internet para acceder a una aplicación de un proveedor de servicios Cloud donde intercambia información de datos o mensajes de control de tráfico. Normalmente utiliza tecnología Wi-Fi o ethernet aunque también mediante conexión celular.

La conectividad desde el Cloud permite al usuario obtener acceso remotamente al equipo y puede tener un potencial soporte debido a actualizaciones que se pueden aplicar.

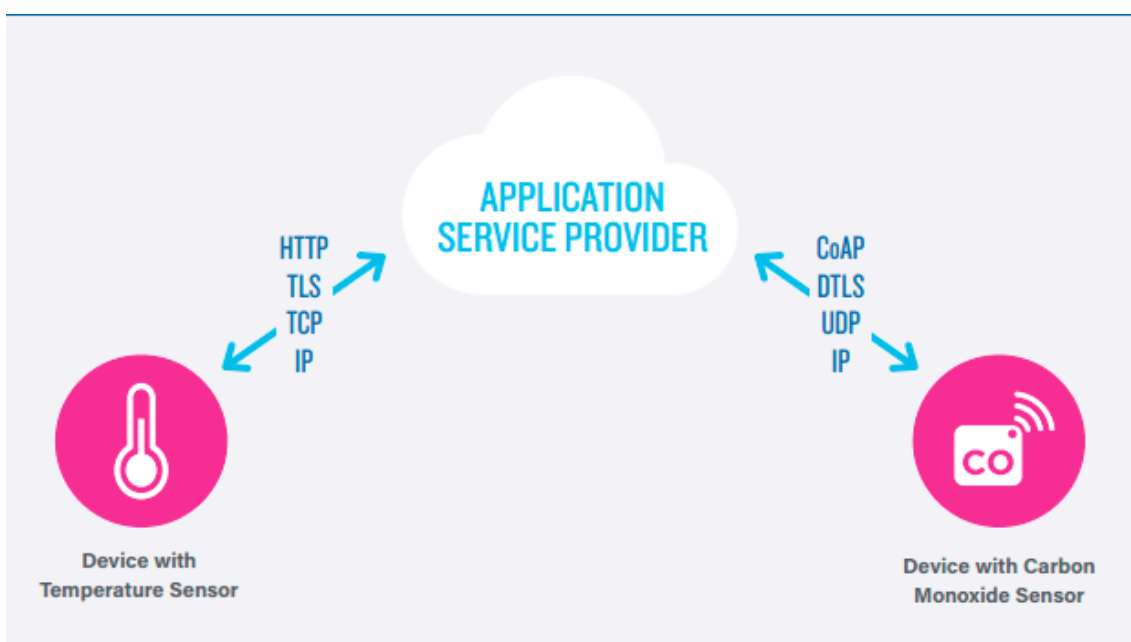


Figura 8. Comunicación equipo a Cloud[8].

- Equipo a Gateway: en este tipo de modelo los equipos IoT básicamente se conectan a un equipo intermedio para acceder a servicios Cloud, normalmente involucra aplicaciones de software en el Gateway local que actúa como intermediario entre el equipo IoT y el Cloud.

Este Gateway puede proveer de servicios de seguridad y otras funcionalidades como traducción de datos y protocolos, pueden ser el puente para la interoperabilidad de equipos que utilizan distintos estándares como por ejemplo Z-Wave y ZigBee.

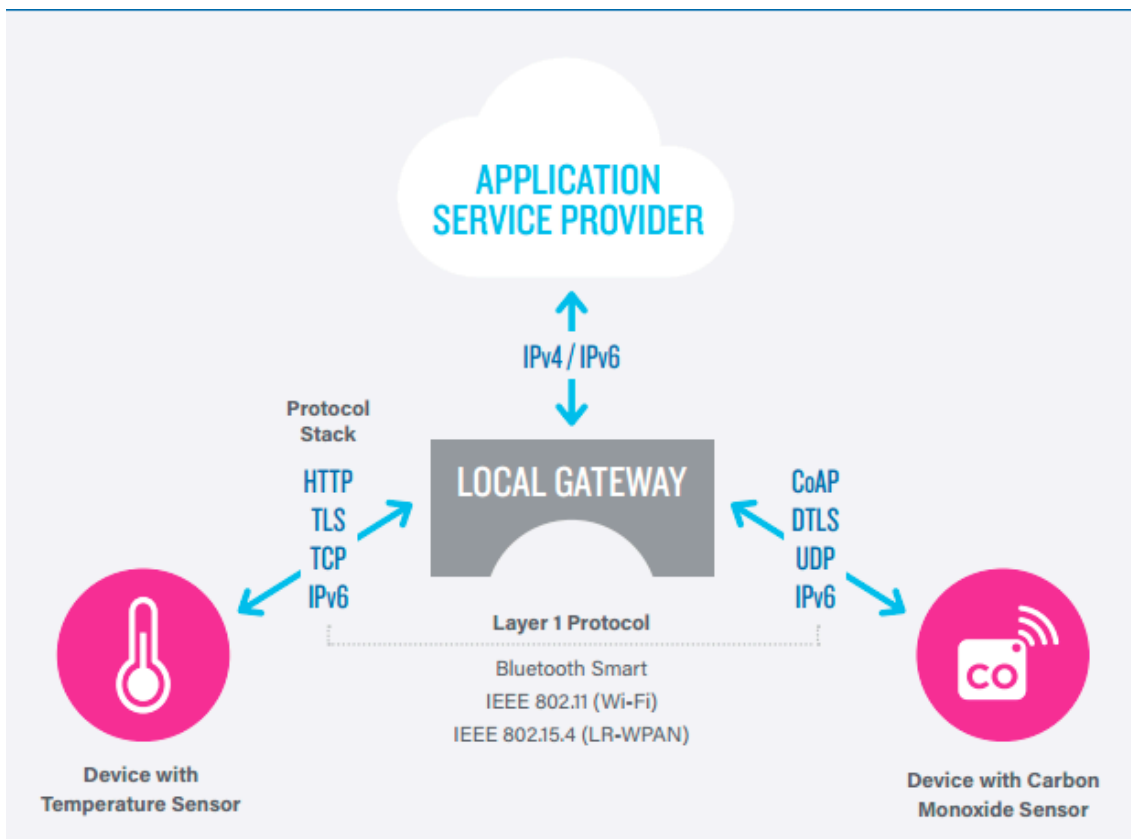


Figura 9. Comunicación equipo a Gateway[8].

- Compartición de datos con Back-End: básicamente extiende el equipo a comunicarse con el Cloud de modo que los sensores de datos y los equipos IoT pueden ser accedidos desde terceras partes autorizadas. Bajo este modelo los usuarios pueden exportar y analizar datos de objetos inteligentes desde un servicio Cloud en combinación con datos de otras fuentes y enviarlos a otros servicios para agregación y análisis.

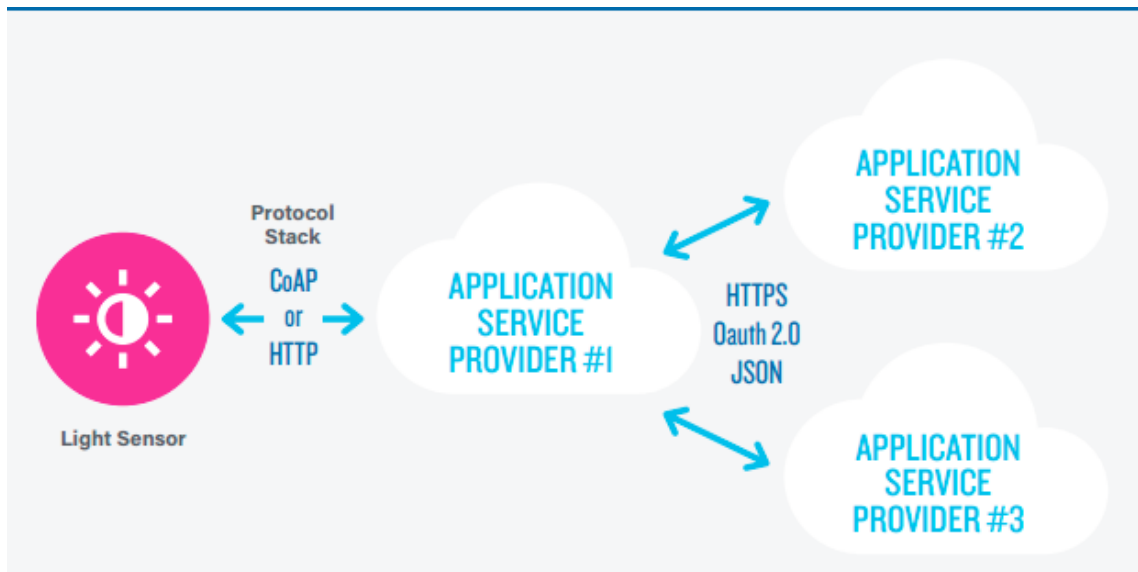


Figura 10. Compartición de datos con Back-End[8].

Aun definiendo estos cuatro modelos de comunicación para IoT no existe un modelo de despliegue claro, depende del caso de uso, hay que considerar como será integrado y como se obtendrá la conexión a Internet.

2.1.4 Campos de aplicación IoT.

El concepto IoT se basa en la interconexión de cualquier objeto con cualquier otro de su alrededor, en los últimos años ha evolucionado tanto que puede abarcar cualquier campo que podamos imaginar, el objetivo es hacer que todos los dispositivos se comuniquen entre sí dotándoles de mayor inteligencia e independientes del ser humano. A continuación, se realiza una descripción de los campos más demandados:

- Smart City[9]: en este campo tanto empresas como instituciones públicas están apostando últimamente, el concepto real es obtener muchos datos para poder facilitar la vida al ciudadano, pueden ser aplicaciones como un alumbrado inteligente de la ciudad para no malgastar energía, aplicaciones de control de tráfico, indicando al usuario la saturación de las vías y rutas alternativas al destino, etc.
- Wearables[10]: se trata de dispositivos de consumo, como pulseras de monitorización que toman datos de nuestro ritmo cardiaco, pasos realizados, etc.
- Smart home[11]: se trata del campo por el que más se ha apostado tanto por parte de empresas como usuarios particulares, se pueden realizar todo tipo de aplicaciones para un hogar inteligente, control de temperatura, control de la iluminación del hogar, subir y bajar persianas automáticamente, sistemas de seguridad, etc. las posibilidades están abiertas a la imaginación del usuario.
- Agricultura[12]: utilizando IoT para realizar un análisis del terreno, su composición y nutrientes, un control del uso de agua y el fertilizante necesario para el campo.

Como se ha visto esto es solo un resumen de una serie de campos de aplicación otros podrían ser para sanidad, agricultura, entornos industriales, automatización, etc.

2.2 IEEE 802.15.4

Este estándar[13] define la capa física y la capa de control de acceso al medio, tiene un bajo consumo de energía, permitir una comunicación con baja transferencia para crear redes LR-WPAN (Low Rate Wireless Personal Area Network).

Las redes que se crean bajo este estándar tienen como objetivo que se auto-organicen y se mantengan en funcionamiento, soporta distintas topologías como en estrella, peer-to-peer o en árbol.

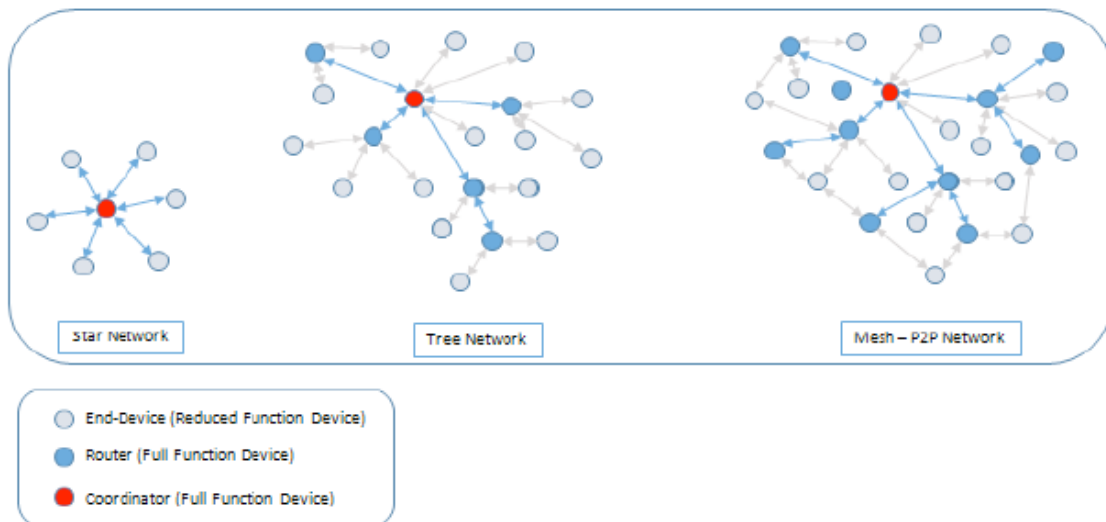


Figura 11. Topologías IEEE 802.15.4[13].

- Capa física: se utiliza espectro ensanchado por secuencia directa (DSSS), dependiendo de la frecuencia se utilizan distintas modulaciones BPSK,ASK, etc. con una sensibilidad definida en el estandar para los transmisores de -85dBm aunque en la práctica suelen tener unos -95dBm. En la siguiente tabla se muestra los distintos rangos de frecuencias y su modulación.

Capa Física	Banda	Parámetros de los datos		
		Vel. bits (kb/s)	Vel. de símbolos (kbaud)	Modulación
868/915 MHz PHY	868.0-868.6 MHz	20	20	BPSK o O-QPSK
	902.0-928 MHz	40	40	BPSK o O-QPSK
2.4 GHz PHY	2.4-4.4845 GHz	250	62.5	16-ary ortogonal

Tabla 2. Rangos frecuenciales.

IEEE 802.15.4 define 27 canales de frecuencia en las distintas bandas, la PHY de 868/915 MHz solo soporta un canal entre los 868 y los 868.6 MHz y otros 10 canales entre los 902.0 y 928 MHz mientras que la PHY de 2.4 GHz soporta los 16 canales restantes entre los 2.4 y 2.4835 GHz.

IEEE 802.15.4 PHY Overview Operating Frequency Bands

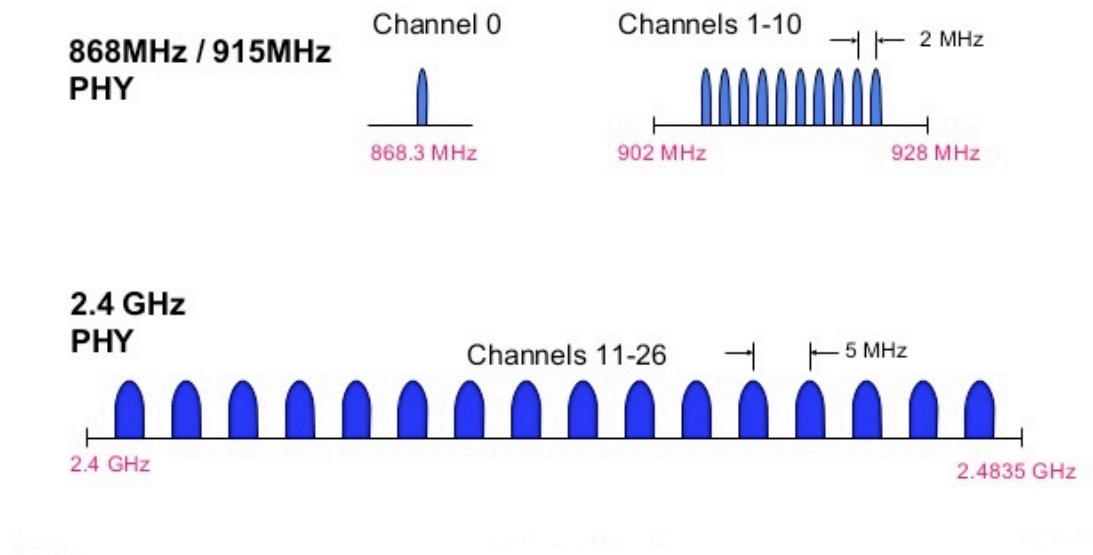


Figura 12. PHY IEEE 802.15.4[13].

- Capa de enlace de datos (DLL): está dividida en dos subcapas MAC (Medium Access Control) y LLC (Logical Link Control) que es común en todos los estándar 802, mientras que la subcapa MAC varía dependiendo del hardware y de su implementación física.

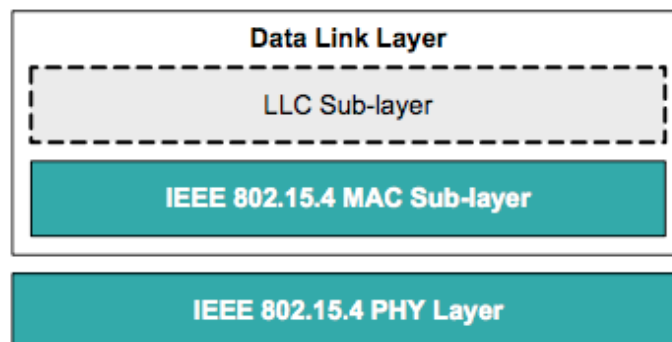


Figura 13. IEEE 802.15.4 Data Link Layer[13].

Las principales funciones de la subcapa MAC son el control y prevención de colisiones mediante CSMA/CA, construcción de topologías, asignación de recursos en tiempo real para obtener una reserva garantizada. Existen dos clases diferentes de acceso al canal[10]:

- Beacon:** en este modo el Coordinador envía periódicamente señales de beacon que contienen información que permite a los nodos sincronizar sus

comunicaciones, normalmente 2 beacon sucesivos marcan el comienzo de una supertrama que contiene 16 timeslots que pueden ser utilizados por los nodos para comunicarse, el tiempo total de esos timeslots se llama CAP (Content Access Period).

Existen timeslots consecutivos llamados GTs (Guaranteed Timeslots), están localizados después del CAP, el tiempo total de GT se denomina Contention Free Period (CFP).

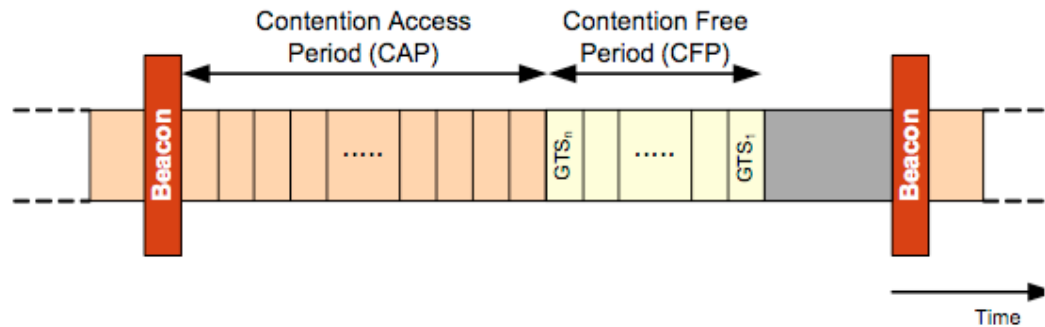


Figura 14. Supertrama[13].

- Sin beacon: en este modo el nodo Coordinador no transmite beacons regularmente, las comunicaciones son en modo asíncrono, un equipo se comunica solo con el coordinador cuando es necesario, lo que permite un ahorro energético importante, este modo es útil en situaciones donde se espera poco tráfico.

Los nodos en IEEE 802.5.14 se definen bajo los siguientes roles:

- Coordinador PAN: solo puede existir uno, debe asignar un PAN ID a la red, encontrar una radio frecuencia adecuada para la operación de red, asignarse una dirección corta, manejar peticiones de otros nodos para unirse a la red.
- Coordinador Local: una red en árbol puede tener uno o más coordinadores locales, cada uno de ellos sirve a sus nodos hijo.
- Dispositivo final: se trata de un nodo que tiene función de entrada/salida pero no de coordinación.
- Full Function Device (FFD): es un nodo que tiene un set completo de servicios IEEE 802.5.14 MAC, pudiendo actuar de coordinador si se diera el caso.
- Reduced Function Device (RFD): este nodo solo posee un set reducido de funcionalidades IEEE 802.5.14, nunca puede actuar como coordinador.

Solo nodos que sean FFD podrán ser el coordinador en la red.

2.2.1 Red IEEE 802.5.14.

Para el setup de la red asumimos una topología en estrella y una red con beacon desactivado.

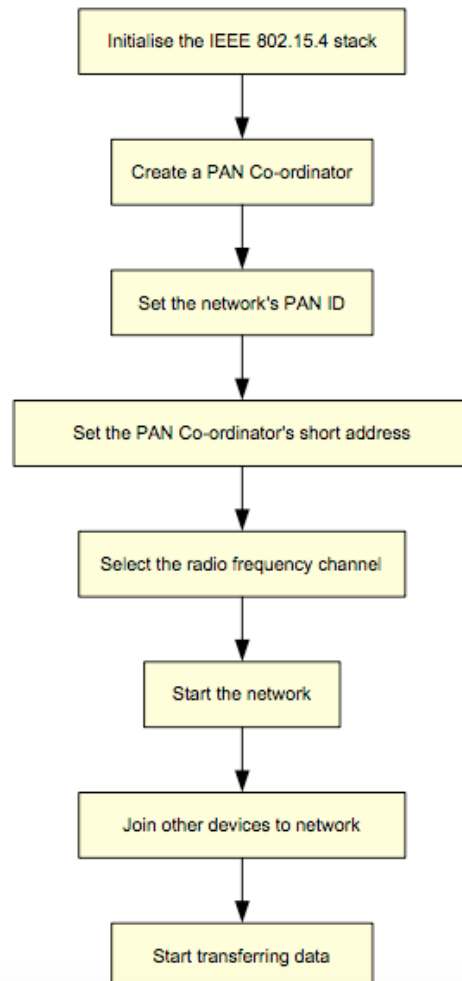


Figura 15. Network Setup[13].

- Inicialización del stack: la capa física y las capas MAC de todo el stack se deben inicializar en todo elemento que quiera formar parte de la red.
- Crear el coordinador PAN: cada red debe tener uno, una de las primeras tareas es seleccionar e inicializar el nodo coordinador.
- Seleccionar el PAN ID y la dirección corta del coordinador: el nodo coordinador debe asignar un ID a la red PAN, este puede ser predeterminado, por otra parte este nodo tiene una dirección de 64bit IEEE (mac) pero debe también tener una dirección de 16bit para poder hacer las comunicaciones más ligeras y eficientes.
- Seleccionar radiofrecuencia: el coordinador debe seleccionar el canal de radiofrecuencia en el que operara dentro del rango frecuencial elegido, puede seleccionarlo mediante un escaneo de detección de energía en el que escanea

los canales hasta encontrar uno válido, se puede programar para solo escanear ciertos canales.

- Iniciando la red: una vez realizados los pasos anteriores el nodo coordinador está a la escucha de solicitudes de otros equipos para unirse a la red.
- Unión de dispositivos a la red: primero los equipos se deben inicializar y luego realizar una solicitud al nodo coordinador, para poder localizar al coordinador el equipo realiza un escaneado activo del canal en el que envía beacons en las frecuencias relevantes, cuando este paquete es detectado por el nodo coordinador este envía otro beacon indicando su presencia.

Una vez detectado envía una solicitud de asociación al coordinador el cuál mira si tiene los recursos necesarios para poder soportar el nuevo equipo y acepta o rechaza la solicitud, si se acepta asigna una dirección de 16bit al nuevo nodo.

La comunicación en IEEE 802.5.14 se basa en un sistema de datos y tramas MAC con ACK opcionales, cuando un nodo envía un mensaje a otro nodo el nodo receptor puede enviar un ACK opcional que solo confirma que ha recibido el mensaje.

Las transferencias de datos pueden ser de la siguiente manera:

- Coordinador a nodo: existen dos métodos, en una topología en estrella los nodos serían el coordinador y equipo final, mientras que en una topología en árbol o mallada puede existir aparte del coordinador y los equipos finales coordinadores locales.

Transmisión directa: el coordinador envía las tramas directamente al nodo final, una vez que han llegado este nodo envía un ACK al coordinador, en este caso el nodo final debe ser capaz de recibir datos, ha de estar siempre activo.

Transmisión indirecta: el coordinador mantiene los datos hasta que esos datos son requeridos por un nodo final, en este caso para obtener los datos el nodo preguntar al coordinador sin los datos están disponibles. Para realizarlo envía una solicitud y el coordinador devuelve un ACK, después el coordinador determina si tiene información para el nodo, si es así la envía, este método es útil cuando el equipo final es de baterías y debe conservar la energía.

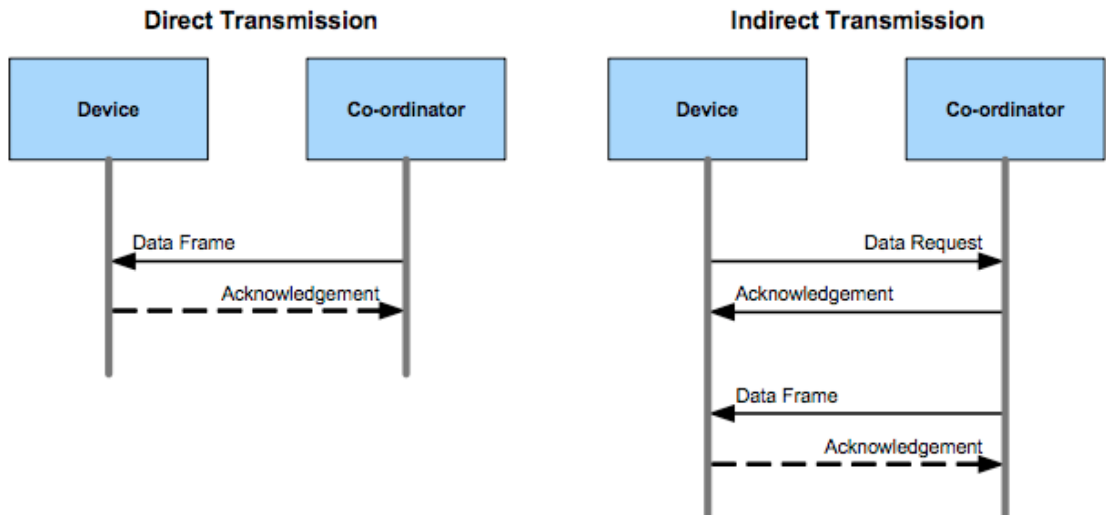


Figura 16. Coordinador a nodo[13].

- Equipo final a coordinador: el equipo final siempre envía las tramas directamente al coordinador, una vez recibidos este puede enviar un ACK de confirmación.
- Coordinador a coordinador: en una topología en estrella o mallada el coordinador siempre envía la información directamente a otro coordinador, una vez recibida puede enviar un ACK de confirmación.

El routing empleado depende directamente de la topología empleada.

- Topología en estrella: todos los mensajes se enrutan a través del coordinador PAN central el cuál es quién aplica el routing.
- Topología en árbol: los mensajes no siempre necesitan ir al nodo coordinador, un mensaje pasa primero desde el nodo que se envía por su nodo padre.

Si el destino es un nodo hijo del mismo padre, el mensaje se envía directamente en cambio sino es esta situación el mensaje se envía por el árbol hacia el siguiente nodo padre, este nodo decide si se ha de enviar al siguiente nodo padre o pertenece a los nodos hijo de su rama.

Un mensaje puede necesitar realizar todo el camino hasta el nodo coordinador antes de que pueda ser enviado a su destino. Se mantiene una tabla de rutas en los nodos coordinadores o se pueden utilizar direccionamientos especiales dependiendo de la posición del árbol donde está un nodo.

- Topología mallada: en esta topología al menos unos nodos son capaces de comunicarse directamente entre ellos, pero no existe una estructura lógica de enrutamiento, aunque se pueden utilizar otros métodos como realizar un broadcast a todos los nodos de la red, otros métodos son el uso de tablas de rutas en nodos de red las cuales son actualizadas cuando se producen cambios

topológicos, pero esto es implementado en capas superiores a IEEE 802.5.14 como por ejemplo en la capa de software de ZigBee.

La comunicación se pasa entre la capa MAC y MAC de usuario (en ambas direcciones) mediante primitivas de servicio, estos mensajes se clasifican de la siguiente manera:

- Request.
- Confirm.
- Indication.
- Response.

La interface MAC opera de la siguiente manera:

1. Un Request es iniciada por el usuario MAC.
2. El Request puede solicitar un Confirm a la capa MAC.
3. Una indicación es iniciada por la capa MAC.
4. La indicación puede solicitar un Response al usuario MAC.

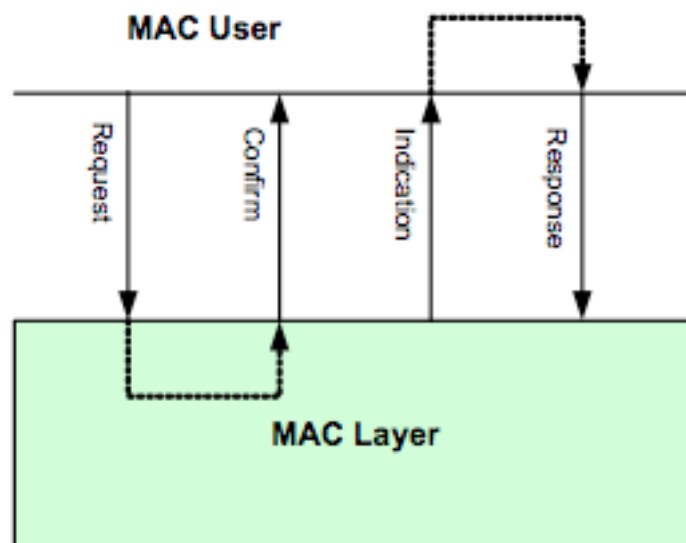


Figura 17. Mecanismo de interface MAC[13].

2.3 IEEE 802.15.4e

Creado por el grupo IEEE802.15 Task Group 4e[14] para poder mejorar y añadir funcionalidades a 802.15.4 MAC, como mejor soporte para el mercado industrial y permitir compatibilidad con modificaciones propuestas por el WPAN, resuelve las siguientes deficiencias de IEEE 802.15.4:

- Consumo energético: en IEEE 802.15.4 los nodos intermedios han de estar siempre activos suponiendo un gran gasto energético, algo no válido para aplicaciones de bajo consumo, IEEE 802.15.4e provee una serie de mejoras MAC y otras de funcionamiento general para solventar este problema.
- Protección Multicamino: IEEE 802.15.4 utiliza un único canal y no cuenta con saltos frecuenciales para obtener una protección a interferencias y desvanecimientos, en cambio IEEE 802.15.4e si introduce esta funcionalidad haciéndolo más adecuado para un entorno IoT.
- Tasa de paquetes baja: IEEE 802.15.4 utiliza CSMA-CA como mecanismo de acceso al medio, esto requiere una sincronización previa entre los nodos de la red, debido a esto y a la aleatoriedad del protocolo la tasa de entrega de paquetes es baja, no siendo adecuado en entornos con muchos nodos o aplicaciones críticas.

Entre las nuevas mejores podemos encontrar:

- Radio Frequency Identification Blink (BLINK): realiza la localización e identificación de personas o ítems.
- Asynchronous multi-channel adaptation (AMCA): su dominio de aplicación es donde se requieren largos despliegues, por ejemplo, monitorización de infraestructura, procesos de automatización o control, etc.
- Deterministic & Synchronous Multi-Channel Extension (DSME): para aplicaciones comerciales e industriales en las se necesita un requerimiento riguroso de tiempo y confiabilidad.
- Low Latency Deterministic Network (LLDN): se utiliza en aplicaciones que necesitan una baja latencia como por ejemplo en control robótico, industrial, etc.
- Time Slotted Channel Hopping (TSCH): el dominio de aplicación es para procesos de automatización, combina acceso a slots por tiempo, comunicación multicanal y salto de canal. La latencia es predecible y tiene un ancho de banda garantizado, los nodos se pueden comunicar al mismo tiempo utilizando distintos canales y mitiga los efectos de interferencias y desvanecimiento por multipath.

Durante un timeslot un nodo envía una trama y otro nodo devuelve un ACK sino se recibe ese ACK en un periodo de tiempo, la retransmisión ha de esperar hasta el siguiente timeslot asignado.

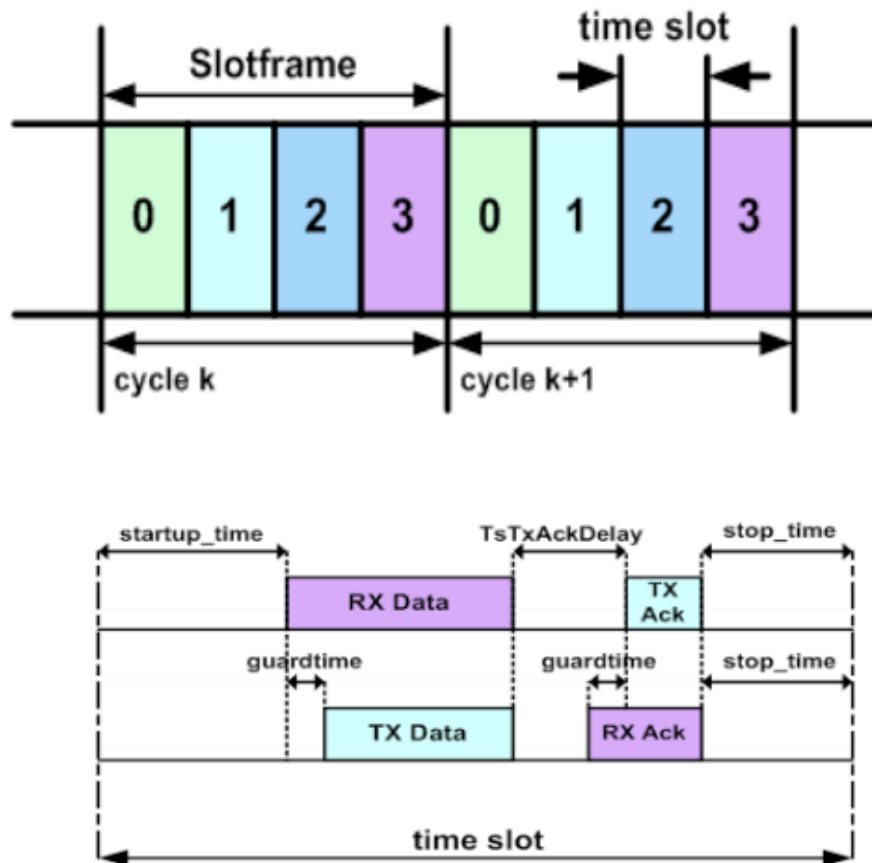


Figura 18. TSCH timeslot[14].

Bajo este estándar se define el protocolo Low Energy (LE) que permite que los dispositivos operen bajo un duty cycle del 1%. Incorpora dos mecanismos para poder garantizar un consumo pequeño de energía Coordinated Sampled Listening (CSL) y Receiver Initiated Transmissions (RIT).

- RIT se utiliza en redes de área personal (PAN) que no utilizan beacons, realiza el envío de tramas datareq periódicamente utilizando CSMA/CA, cada vez que se envía una de estas tramas se realiza una escucha del canal para recibir transmisiones, el emisor está a la espera de recibir una de estas tramas antes de enviar su información.

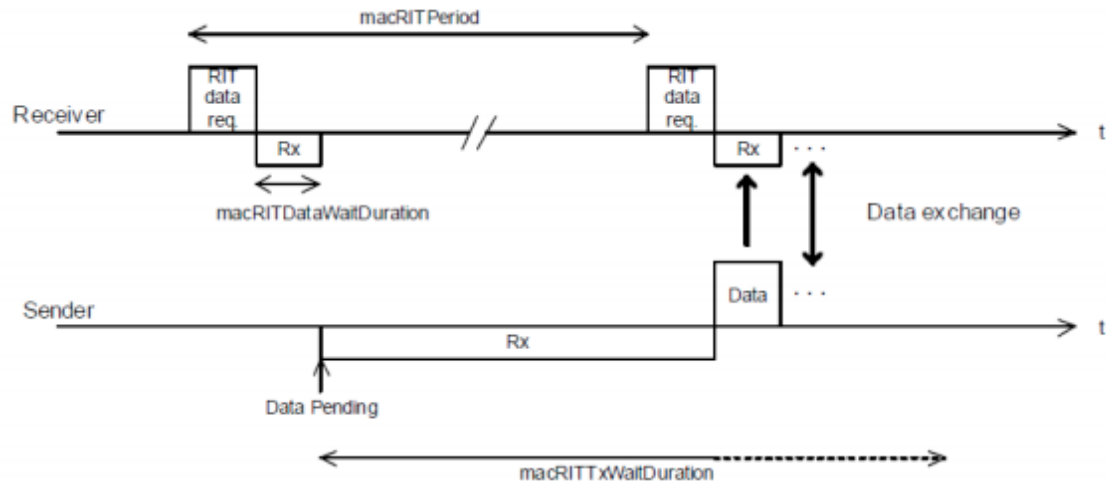


Figura 19. IEEE 802.15.4e RIT[14].

- CSL es un mecanismo para el ahorro energético, el receptor realiza una escucha periódica del canal y comprueba si existe alguna transmisión pendiente. Si está realizando una escucha y recibe una trama wakeup este mecanismo desactiva el receptor durante el denominado Rendezvous Time (RZTime), este tiempo viene incluido dentro de la trama wakeup e indica el tiempo entre el final de la trama wakeup y el principio de la trama de datos por lo que el receptor sabrá cuando volver a estar operativo para recibir los datos de un emisor.

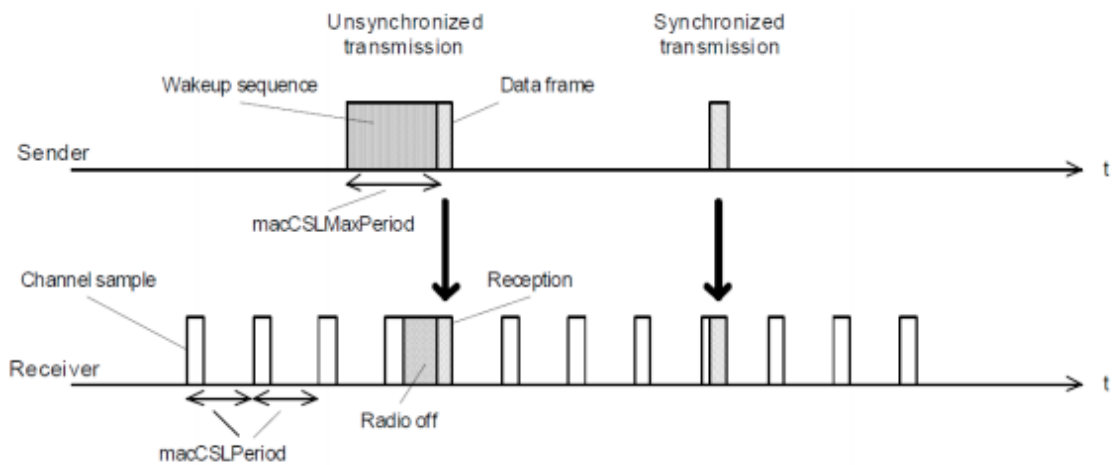


Figura 20. IEEE 802.15.4e CSL[14].

2.4 CoAP.

Constrained Application Protocol (CoAP) publicado como RFC7252[15] es un protocolo de transferencia web para utilizar en nodos y redes restringidas, permitiendo un intercambio de información entre cliente y servidor mediante HTTP.

El modelo de interacción de CoAP es similar a cliente/servidor http, sin embargo las interacciones M2M resultan en una implementación CoAP actuando ambos como cliente/servidor.

CoAP realiza estos intercambios de forma asíncrona sobre un protocolo de transporte de datagramas (UDP) y consta de 4 tipos distintos de mensajes:

- Confirmable (CON): cuando un mensaje que quiere definir como confiable este es marcado como CON, el mensaje se retransmite con un timeout por defecto y un back-off entre retransmisiones hasta que el receptor envía un ACK.
- Non-confirmable (NON): cuando un mensaje no necesita ser confiable se marca como NON, no requieren de ACK por parte de receptor, pero tienen un mensaje ID para detección de duplicados.
- Acknowledgement: cuando se envía un mensaje CON el receptor devuelve un ACK con el mismo ID del mensaje.
- Reset: si un receptor no puede procesar un mensaje CON este devuelve al emisor un Reset.

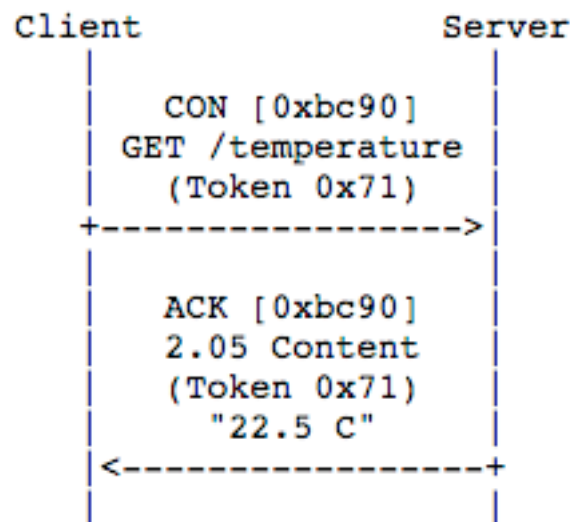


Figura 21. Mensaje CON CoAP[15].

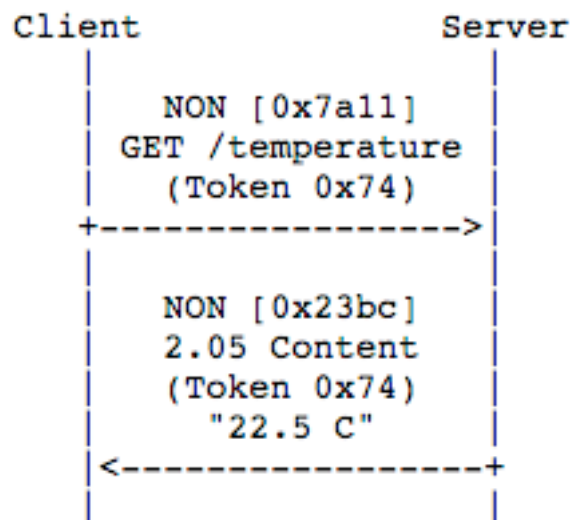


Figura 22. Mensaje NON CoAP[15].

CoAP se basa en el intercambio de mensajes compactos transportados por UDP, aunque también puede utilizarse sobre Datagram Transport Layer Security (DTLS) u otros protocolos de transporte como SMS, TCP o SCTP. Los mensajes están codificados en formato binario simple. En la siguiente imagen podemos ver la estructura:

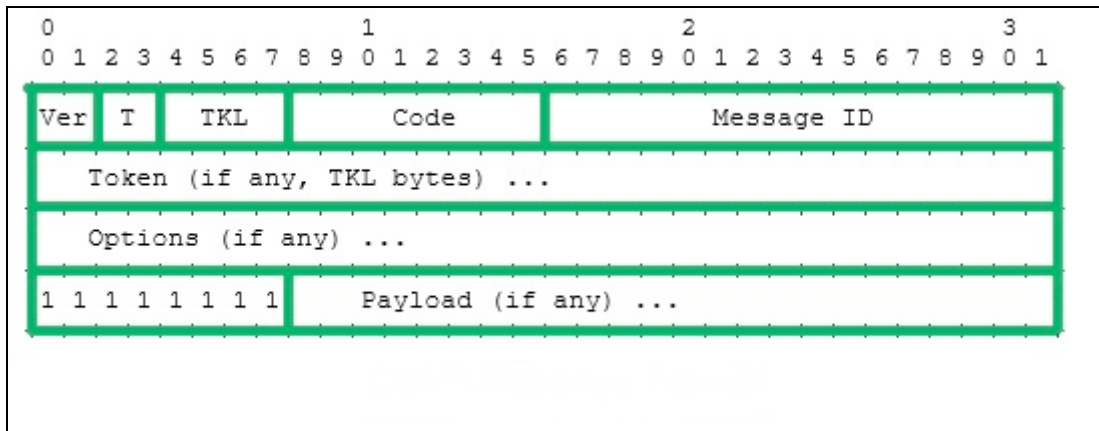


Figura 23. Formato mensaje CoAP[16].

- Version (Ver): indica la versión de CoAP mediante 2 bits.
- Type (T): dependiendo del valor puede ser (0) Confirmable, (1) Non-confirmable, (2) ACK, (3) Reset.
- Token Length (TKL): entero de 4 bits que indica la longitud del campo de Token de longitud variable (0-8 bits), las longitudes entre 9-15 no se deben enviar y han de ser procesadas como mensaje de error.
- Code: entero de 8 bits dividido en una clase de 3 bits (los más significantes) y otra de 5 bits (los menos significantes).
- Message ID: campo de 16 bits utilizado para detectar mensajes duplicados y para emparejar mensajes del tipo ACK/Reset a mensajes CON/NON
- Opciones: aplicar distintas opciones al mensaje, ETag Uri-port, etc.
- Payload: el contenido propio del mensaje.

CoAP maneja los siguientes métodos:

- GET: recupera una representación para la información que corresponde al origen identificado por la URI solicitada. Devuelve un 2.05 (Content) o 2.03 (Valid) si el GET ha sido satisfactorio.
- POST: la función actual de POST es determinada por el servidor origen y dependiente del recurso objetivo. Si el recurso ha sido creado en el servidor, la respuesta 2.01 (Created), si POST ha sido satisfactorio, pero no resulta en la creación de un nuevo recurso en el servidor la respuesta será 2.04 (Changed) y si la respuesta es correcta y se realiza el borrado del recurso objetivo devolverá 2.02 (Deleted).

- PUT: este método define que el recurso identificado en la URI debe ser actualizado o creado, si el recurso existe debe devolver un 2.04 (Changed), si no existe se debe crear el nuevo recurso devolver un 2.01 (Created).
- DELETE: solicita que el recurso identificado por la URI sea borrado, se debe generar una respuesta 2.02 (Deleted).

CoAP Status Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
2.31	Continue
4.00	Bad Request
4.01	Unauthorized
4.02	Bad option
4.03	Forbidden
4.04	Not Found
4.05	Method not Allowed
4.06	Not Acceptable
4.08	Request Entity Incomplete
4.12	Precondition Failed
4.13	Request Entity too large
4.15	Unsupported Content Format
5.00	Internal Server Error
5.01	Not implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxing Not Supported

Tabla 3. CoAP Status Code.

2.3 Fog computing.

Este sistema también conocido como fog networking o fogging[17] es un sistema descentralizado de infraestructura de computación en el que los datos son tratados, almacenados y las aplicaciones son distribuidas en el lugar más eficiente entre el origen de los datos y el cloud.

El principal objetivo es mejorar la eficiencia y reducir la cantidad de información que se transporta al cloud para procesar, analizar y almacenar, por otra parte, al solo enviar los

datos requeridos ya procesados se realiza un incremento de la seguridad en la información transportada.

En este tipo de computación nos encontramos con tres elementos distintos:

- Dispositivos edge: pueden ser toda clase de sistemas de sensores o aplicativos que nos devuelvan información solicitada dependiendo del sistema configurado.
- Dispositivos fog: se encargan de realizar la función de puente entre nuestra red local de dispositivos donde recolectamos información y el cloud, también se encargan de realizar el procesamiento de la información obtenida para enviar solo lo necesario a tratar en el cloud.
- Cloud: se trata del aplicativo en la nube a donde enviamos nuestros datos para procesar, almacenar o tratar, ya

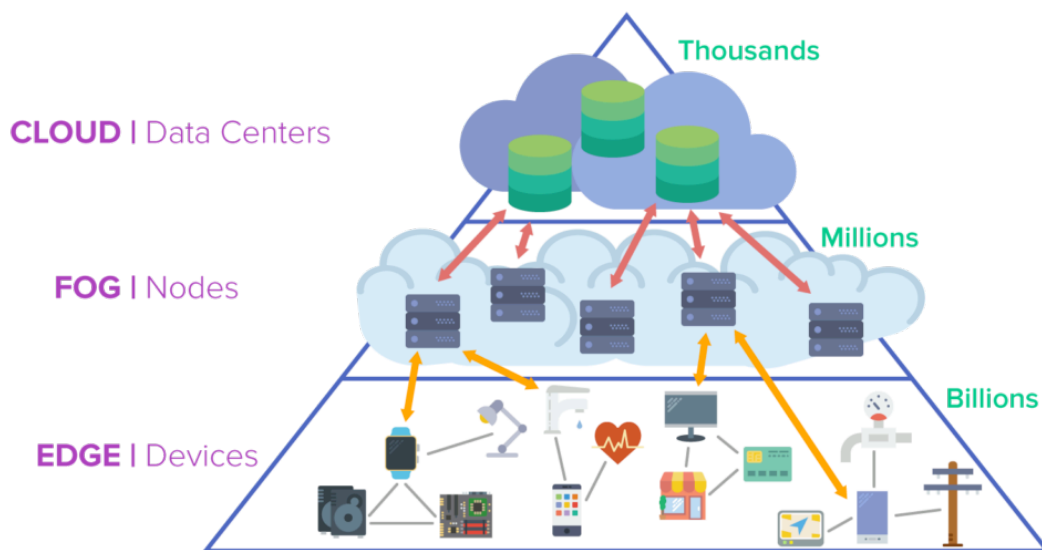


Figura 24. Dispositivos en Fog computing[18].

Entre los beneficios podemos encontrar los siguientes:

- Mejor seguridad: proteger los nodos fog utilizando las mismas políticas, controles y procedimientos que se utilizan en otra parte del entorno, usar la misma seguridad física y lógica.
- Gastos de operativa menor: al procesar los datos en local se realiza un ahorro en el consumo de ancho de banda requerido para enviar la información a analizar al cloud.
- Control de privacidad: se analizan los datos sensibles localmente sin tener que enviarlos al cloud, podemos monitorizar, vigilar y controlar los equipos locales que analizan los datos antes de enviarlos.
- Mejora de la agilidad para negocio: los desarrolladores pueden desarrollar y desplegar aplicaciones cuando sea necesario rápidamente.

La arquitectura de este sistema es horizontal [19] da soporte a muchos dominios de aplicación u de industria verticales dotando de inteligencia y servicios a los usuarios y a negocio.

Continuación de servicios del cloud al elemento thing, habilita que los servicios y aplicaciones estén distribuidos lo más cercano a nuestros elementos que reportar información y datos.

Se trata de un sistema de niveles extendido por los nodos, a través de los nodos fog y el cloud, pasando por múltiples capas de protocolos.

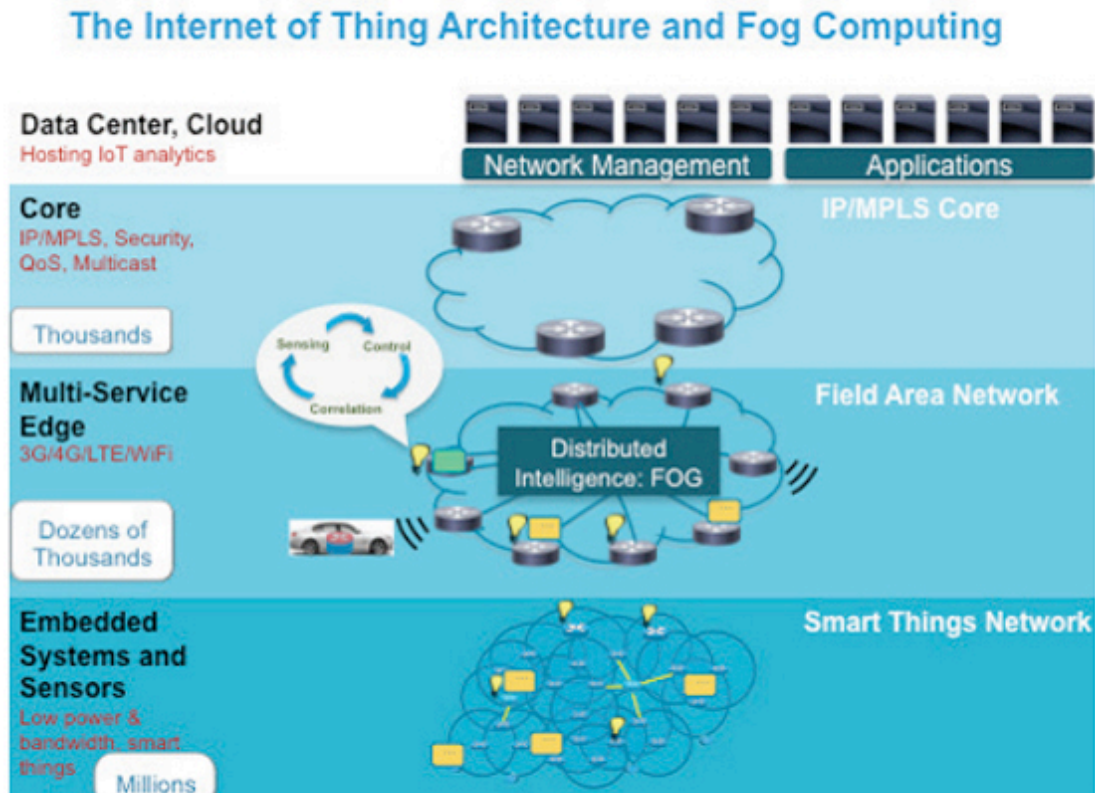


Figura 25. Arquitectura Fog computing[20].

2.6 Cloud computing.

El término Cloud Computing es un paradigma IT que habilita un acceso a pools compartidas donde existen recursos configurables, así como recursos de alto nivel que pueden ser rápidamente provisionados con un mínimo de esfuerzo normalmente sobre Internet.

Comenzó[21] en proveedores de servicios de Internet a gran escala como Amazon AWS, Google, Microsoft, etc. Entre todos ellos surgió una arquitectura de servicios de recursos con distribución horizontal que son introducidos de manera virtual.

Los Cloud de third-parties proporcionan a las organizaciones el poder centrarse en el Core de su negocio en vez de tener que gastar sumas elevadas en recursos de infraestructura de computación y mantenimiento.

Este modelo de prestación de servicios permite al usuario acceder a un catálogo de servicios estándar y utilizarlos para sus necesidades de negocio, de una forma flexible y adaptativa ya que en casos de picos de trabajo puede pagar únicamente por el consumo efectuado.

Este modelo de computación ofrece los siguientes servicios:

- **Software as a Service (SaaS):** permite a los usuarios conectarse a aplicaciones basadas en la nube a través de Internet y utilizarlas (e-mail, ofimática, etc.), es un modelo de distribución de software donde el soporte lógico y los datos están alojados en servidores de una compañía proveedora del servicio. Esta empresa se encarga del mantenimiento, operación y soporte de software utilizado por el cliente.
- **Platform as a Service (PaaS):** esta clase dota de una plataforma a los clientes permitiendo desarrollar, inicializar y gestionar aplicaciones sin la complejidad de crear y mantener la infraestructura asociada al desarrollo. El consumidor controla el despliegue de software con un mínimo de opciones, servidores, almacenamiento, sistemas operativos, middleware (.Net, Java, etc.), bases de datos y otros servicios.
- **Infrastructure as a Service (IaaS):** en esta capa se hace entrega de almacenamiento o capacidad de cómputo, así como servicios de red, servicios de backup, hypervisores (Xen, Oracle, etc.). El consumidor no tiene control de la capa de infraestructura, pero controla los sistemas operativos, el almacenamiento y las aplicaciones desplegadas.



Figura 26. Servicios Cloud Computing.

Aparte de los distintos tipos de cloud computing nos ofrece distintas clases de nube:

- Pública: el mantenimiento y la gestión se realiza mediante terceras personas que no tienen vinculación con la organización. Los datos y los procesos de distintos clientes se mezclan en los servidores. El usuario final no conoce que trabajos e información de otros clientes están corriendo en el mismo servidor. Tanto las aplicaciones como el almacenamiento y otros recursos son propiedad del proveedor del servicio, así como la infraestructura en sus centros de datos, este tipo de servicio es accesible vía Internet.
- Privada: están bajo una infraestructura bajo demanda, tiene una gestión para un solo cliente que tiene el control de qué aplicaciones deben ejecutarse y donde. Tienen la propiedad de los servidores, la red y demás necesidades y son los encargados de autorizar quién puede o no acceder a dicha nube. Tiene la ventaja de que mantienen la privacidad de su información y permite realizar una unificación para el acceso de aplicaciones de tipo corporativo.
- Híbridas: es un modelo que combina la nube pública y la privada, un usuario es propietario de determinadas partes a las que sólo él puede acceder pero comparte otras de una manera controlada. Ofrecen aprovisionamiento bajo demanda pero tienen la complejidad de cómo distribuir las aplicaciones a través de los distintos ambientes.

En la siguiente tabla resumen se ve una comparativa de las distintas posibilidades de nube:

Nube	Usuarios	Recursos	Control	Proveedores
Privada	Un solo cliente	Hardware local	Control directo infraestructura	HPE, VMWare, IBM, EMC, OpenStack.
Pública	Diversos clientes	Pago por recursos	Infraestructura del proveedor	AWS, Azure, Google Cloud.
Híbrida	Diversos clientes	Ambas nubes	Control del proveedor y usuario	Proveedores privados y públicos.

Tabla 4. Modelos nube Cloud computing.

Todas las soluciones tienen sus ventajas y desventajas en temas de privacidad, recursos, coste, etc. La elección de una u otra solución dependerá del tipo de servicio requerido por el usuario, modelo de negocio, privacidad, coste, etc.

2.7 Comparación Cloud computing – Fog computing.

Ambas soluciones de computación abarcan distintos modelos, clases y funcionalidades, que quedan reflejadas en los capítulos anteriores, pero haciendo referencia a IoT ya no solo encontramos diferencias en los modelos de servicio sino al tratamiento de la información y la calidad de servicio.

Los dos sistemas nos permiten crear un diseño para IoT cada uno con sus ventajas y desventajas, a nivel de QoS y aplicaciones en tiempo real las diferencias serían:

Requerimientos	Cloud Computing	Fog Computing
Latencia	Alta	Baja
Jitter	Alta	Muy bajo
Localización del servicio	Internet	Final red local
Distancia client/server	Múltiples saltos	Uno
Seguridad	Sin definición	Puede ser definida
Ataque datos	Alta probabilidad	Poca probabilidad
Importancia localización	No	Si
Geo-distribución	Centralizada	Distribuida
Nº nodos	Pocos	Amplia
Movilidad	Limitada	Soportada
RTI	Soportado	Soportado
Última milla	Línea arrendada	Wireless

Tabla 5. Qos Cloud computing vs Fog computing.

Al observar la tabla anterior se puede ver que en un entorno IoT donde tendremos una red compuesta por distintas clases de sensores, actuadores etc. prima unos requerimientos bajos de latencia para poder obtener una rápida interacción con nuestro entorno, la mejor calidad de servicio sería la proporcionada por fog computing como podemos ver en la siguiente comparativa:

Cloud Computing	Fog Computing
Los datos y aplicaciones son procesados en el cloud, lo cual consume mucho tiempo para largas cadenas de datos.	Opera en el final de la red, tiene menor consumo de tiempo al no tener que ir a través de Internet.
Problema de ancho de banda, cada bit ha de ser enviado al cloud.	Menos demanda de ancho de banda, cada bit va a puntos de acceso.
Alto tiempo de respuesta y problemas de escalabilidad al depender de servidores remotos.	Desplegando pequeños servidores bajo la visibilidad de los usuarios se evita delay y problemas de escalabilidad.

Tabla 6. Delay Cloud computing vs Fog computing.

A medida que el número de dispositivos conectados aumenta la cantidad de datos generados crece exponencialmente, en un modelo de cloud computing se podría estar creando datacenters específicos durante un tiempo pero a la larga esta solución no es práctica debido a un gran incremento de costes, consumo energético, tiempos de respuesta, interrupciones de servicio, escalabilidad del sistema, etc.

Como ejemplos de elementos en los que fog computing tiene un rol vital tenemos[22]:

- Vehículos conectados (CV): esta distribución nos muestra una gran conectividad e interacción entre vehículos, ya sea coche a coche, coche a punto de acceso (Wi-Fi, 3G, luces de tráfico inteligentes, etc.).

- Sensores Wireless y actuadores de red (WSAN): los nodos sensores Wireless fueron diseñados para operar en bajo consumo para aumentar la vida de las baterías, la mayoría de estos equipos requieren muchísimo menos ancho de banda, menos energía, capacidad lenta de procesamiento operando como orígenes de un colector de datos en modo unidireccional.
- Redes definidas por software (SDN): el concepto de SDN con fogging determina los mayores problemas en redes vehiculares, conectividad irregular, colisiones, pérdidas de paquetes mediante la extensión vehículo a vehículo con vehículo a infraestructura de comunicación y control unificado.

Como conclusión podemos reflejar que fog computing desempeña mejor las funciones requeridas y demandadas para IoT que cloud computing, pero no puede reemplazar totalmente este tipo de computación ya que siempre será preferible para el procesado de entornos con grandes procesados de datos.

Se puede decir que tanto fog computing como cloud computing se complementan uno a otro cada uno con sus ventajas y desventajas, fog elimina el paradigma de redes emergentes que requieren un procesado rápido de datos con poco retardo y jitter mientras que cloud aporta su alta cantidad de computación bajo un modelo de pago bajo demanda.

2.8 Sistemas operativos para IoT.

IoT abarca un ancho rango de máquinas desde sensores con microcontroladores de 8 bits a equipos con microprocesadores equivalentes a los de un smartphone. Ninguno de los sistemas operativos tradicionales que actualmente están funcionando en host conectados a Internet o sistemas operativos típicos para redes sensoriales son capaces de cumplir los requerimientos de tan amplio espectro de equipos.

A continuación, se realiza una breve descripción de algunos de los OS para IoT y una explicación detallada del sistema que se utiliza en este proyecto OpenWSN en el siguiente capítulo.

- Contiki [23]: diseñado en 2002 por Adam Dunkels [19], se trata de un sistema operativo que funciona en microcontroladores de bajo consumo y hace posible el desarrollo de aplicaciones para hacer un uso más eficiente del hardware.

Entre sus características están: ipv4, ipv6, tcp/ip, 6lowpan, CoAP, RPL, etc. además de su simulador Cooja donde se pueden simular utilizando el firmware de los dispositivos físicos una red IoT.

- RIOT [24]: soporta la mayoría de dispositivos de bajo consumo para IoT y microcontroladores de 32, 16 y 8 bit, tiene programación estándar bajo C y C++, el desarrollo se puede realizar en Linux o Mac OS y desplegar directamente a sistemas embebidos.
- Apache Mynewt [25]: similar a RIOT, se trata de un sistema en tiempo real para IoT, el código está distribuido bajo licencia Apache 2.0, es capaz de funcionar en equipos limitados con un mínimo de 8Kb de RAM y 64Kb de ROM, soporta Bluetooth LE, LoraWAN y Wi-Fi.

2.8.1 OpenWSN.

OpenWSN [26] es un sistema que sirve como repositorio para implementaciones open-source basadas en el stack de protocolos del estándar IoT, en el que se puede utilizar una gran variedad de hardware y software.

Bajo el stack de protocolos que maneja está IEEE802.15.4e, IETF 6TiSCH, 6LoWPAN, ROLL, CoAP, RPL, etc.

application	CoAP, HTTP
transport	UDP, TCP
IP/routing	IETF RPL
adaptation	IETF 6LoWPAN
medium access	IEEE802.15.4e
phy	IEEE802.15.4-2006

Figura 27. Stack protocolos OpenWSN[22].

2.8.2 OpenSim.

OpenSim [27] permite simular una red OpenWSN sin necesidad de los equipos físicos. El comportamiento es el mismo que en una red con hardware real solo que el firmware está funcionando en el ordenador o dispositivo donde se compile en vez de en los endpoints.

La compilación del firmware se realiza mediante una extensión Python y creando una instancia que resulta en una clase para cada mote emulado, interactúa con bus de eventos de la misma manera que la instancia motprobe realiza con los sistemas físicos.

Opensim puede interactuar con OpenVisualizer y comunicar nuestros nodos simulados con Internet.

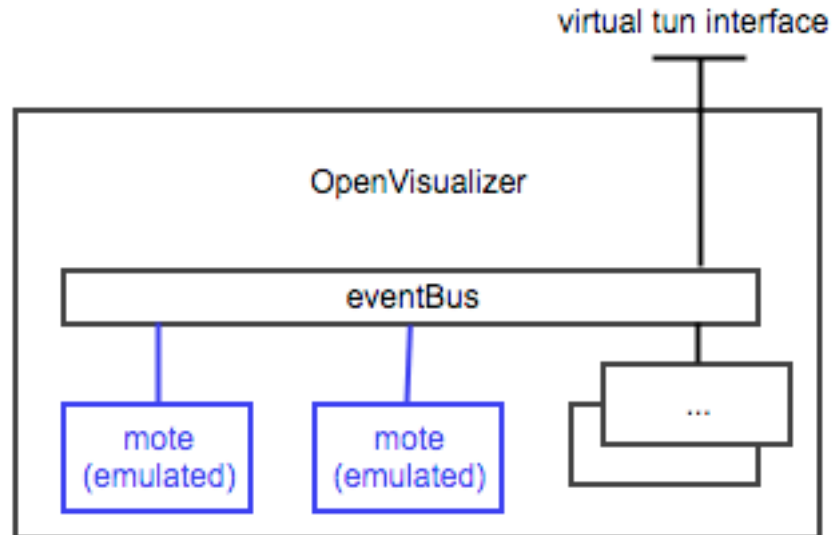


Figura 28. Red OpenWSN simulada[27].

2.8.3 OpenVisualizer.

Se trata de la herramienta principal para conectar nuestra red OpenWSN a Internet, es un entorno de visualización y debugging para los mote conector a la red OpenWSN.

Podemos obtener las siguientes funciones una vez inicializado y conectados los mote:

- Definir un nodo como DAG root, nodo coordinador en modo bridge (no alcanzable por icmp).
- Calculo de routing (RPL).
- Tabla interna de vecinos, con direcciones mac, sensibilidad, etc.
- Permite funcionar con motes físicos o simulados.
- Simulación de topología.
- Debug comunicación entre motes y el DAG root.

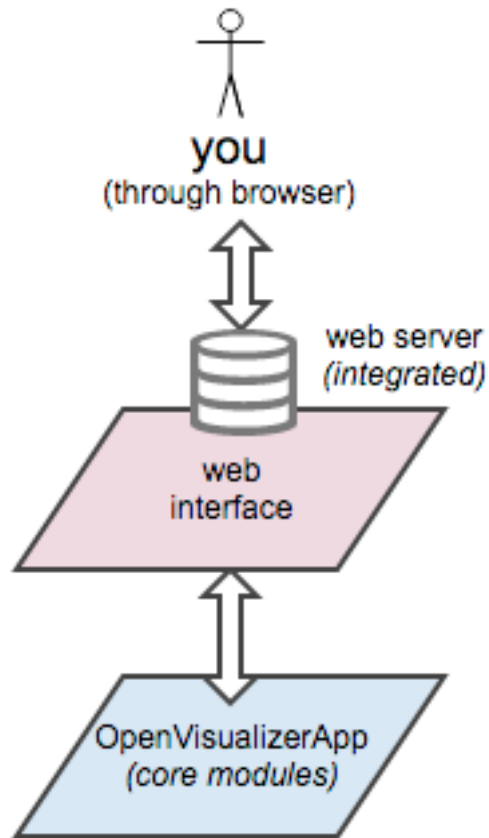


Figura 29. OVS vía web[23].

2.8.4 Librería Python CoAP.

Esta librería se implementa para poder soportar CoAP dentro de OpenWSN, se realiza una implementación cliente servidor, las funciones a las que da soporte [28] se detallan en la siguiente tabla:

Características	Estado
CoAP server capability	Soportado
CoAP client capability	Soportado
CONFirmable messaging	Soportado
NON confirmable messaging	Soportado
Timeout and retry	Soportado
Concurrent requests	Soportado
Caching	No soportado
Proxying	No soportado
DTLS security	No soportado
Multicast CoAP	No soportado

Tabla 7. Características librería CoAP.

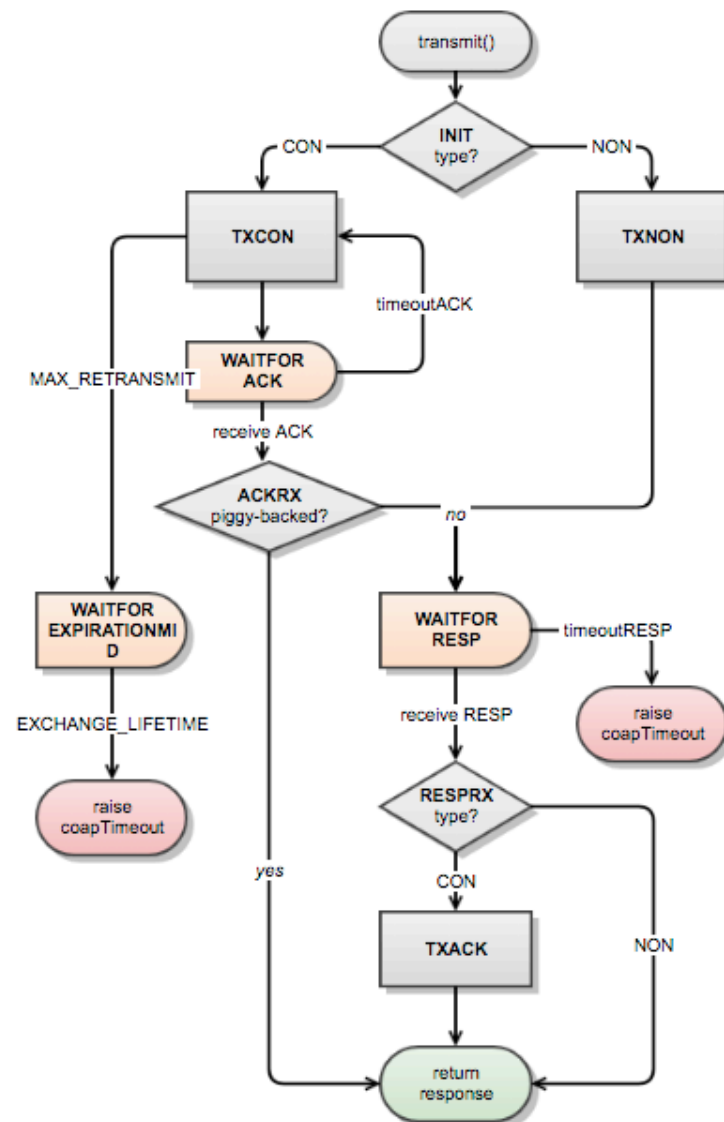


Figura 30. Estados Python CoAP[28].

2.9 Plataformas en la nube.

Una plataforma en la nube es la capa superior donde se centralizan los servicios de la estructura IoT, el objetivo de esta plataforma es la de ofrecer un alto valor añadido, donde el usuario pueda ver con claridad el salto en calidad de sistema que se obtiene al adoptar una plataforma IoT.

Ya sea para realizar una recopilación de datos sin explotar de dispositivos o sensores y utilizar distintas herramientas de bases de datos (sql o nosql) para el posterior análisis, obtención de estadísticas en tiempo real, actuaciones sobre nuestra plataforma IoT, etc.

- Dweet.io[29]: sin configuración previa ni registro, los envíos de información son públicos y accesibles para todo el mundo que conozca la dirección de envío de nuestros datos, se puede realizar el envío privado a una dirección reservada por nosotros a un coste de 1,99\$, los datos se almacenan durante 24 horas con un envío máximo de 5 mensajes (cada mensaje soporta hasta 2000 caracteres).

- Freeboard[30]: realiza una consulta en dweet.io de nuestros datos, y los muestra en una interface gráfica configurable, se ha de especificar la naturaleza de los datos, darles nombre y elegir el tipo de visualización, el servicio gratuito al igual que la anterior plataforma presenta nuestra información en modo público, para tener un servicio privado la plataforma dispone de distintos servicios de pago dependiendo del número de dashboards y etiquetas privadas deseamos tener.
- Amazon Web Services [31]: se trata de una plataforma Cloud que ofrece todas las vertientes, IaaS, SaaS y PaaS. Para el servicio IoT cuenta con una plataforma denominada AWS IoT que permite conectar nuestros dispositivos con la nube mediante protocolos como HTTP, AMQP o MQTT, puede dotarse a la conexión de autenticación para dotar de seguridad la transferencia de datos, uso de base de datos no relacional (DynamoDB) para el tratamiento de la información así como interacción entre nuestros distintos dispositivos, sistemas de alerta, actuadores, etc. Tiene cuenta gratuita durante 1 año pero limitando los recursos a utilizar.
- Thethings.io [32]: se trata de una plataforma Cloud específica, con una conexión sencilla utilizando para ello un API específico. Dentro de la web se crea un objeto que lleva asociado un token identificativo, el cuál se utiliza en dicha API de interconexión para el envío de datos. Provee sistema de dashboard, Jobs, triggers, tiene solución gratuita durante 14 días con un número limitado de envío de datos.

3. Diseño e implementación del sistema.

En este proyecto se quiere realizar una implementación de una red de área personal donde estarán localizados nuestros mote obteniendo información mediante sus sensores, esta información será recogida por el DAGroot es una Raspberry pi, una vez los datos sean leídos estos serán procesados de distintas maneras, una mediante fog computing es decir todo el procesado en la propia Raspberry pi y luego una subida de datos al cloud para realizar una representación de los mismos.

La simulación de la red Open-WSN será una topología en estrella donde cada mote enviará la información recogida al DAGroot siendo este el único punto de comunicación de los mote.

Por otra parte se realizará un procesamiento en el Cloud, los datos una vez obtenidos no son tratados en nuestro Gateway sino que se envían directamente a la plataforma en la nube donde mediante scripts realizan distintas funcionalidades.

El objetivo es poder comparar ambos tipos de procesamiento en términos de velocidad, fiabilidad, eficiencia y aplicación. Poder ver en qué ámbito es más útil un sistema u otro.

3.1 Arquitectura del sistema.

La arquitectura se divide en 3 partes, la correspondiente a la red OpenWSN formada por los OpenMote y la Raspberry pi, la parte de procesado en Raspberry pi donde estarán los scripts de Python para el tratamiento de los datos capturados y la comunicación entre nuestro sistema y la plataforma Thethings.io.

- Red OpenWSN: se trata de una red ipv6 en estrella en la cual se crea una interface túnel con direccionamiento bbbb:0:0:0:1 mediante la cual se hará la comunicación con los OpenMote que tendrán su propia dirección ipv6 formada por su dirección MAC.

Los dispositivos OpenMote tendrán el programa en lenguaje C con las acciones a realizar dependiendo de la petición que obtengan por CoAP, GET, PUT, PUSH ya sea a sus sensores integrados o a actuadores como un relé u otro tipo de sensores.

- Raspberry pi: es el punto central del sistema, tendrá conectado otro OpenMote en modo bridge siendo el DAGRoot para poder comunicarse con los demás mote.

Desde este equipo se realizará la compilación de los OpenMote con su programa en C, además de contar con los ficheros en Python que realizarán las funciones de adquisición y tratamiento de los datos de los sensores. También contendrá una base de datos donde se almacenará la información recogida.

La última función es contar con un script utilizando la api de Thethings.io para poder subir los datos al Cloud.

- Thethings.io: recibirá los datos recogidos y ya tratados por los scripts de la Raspberry pi, aquí se realizarán distintas gráficas de los datos obtenidos.

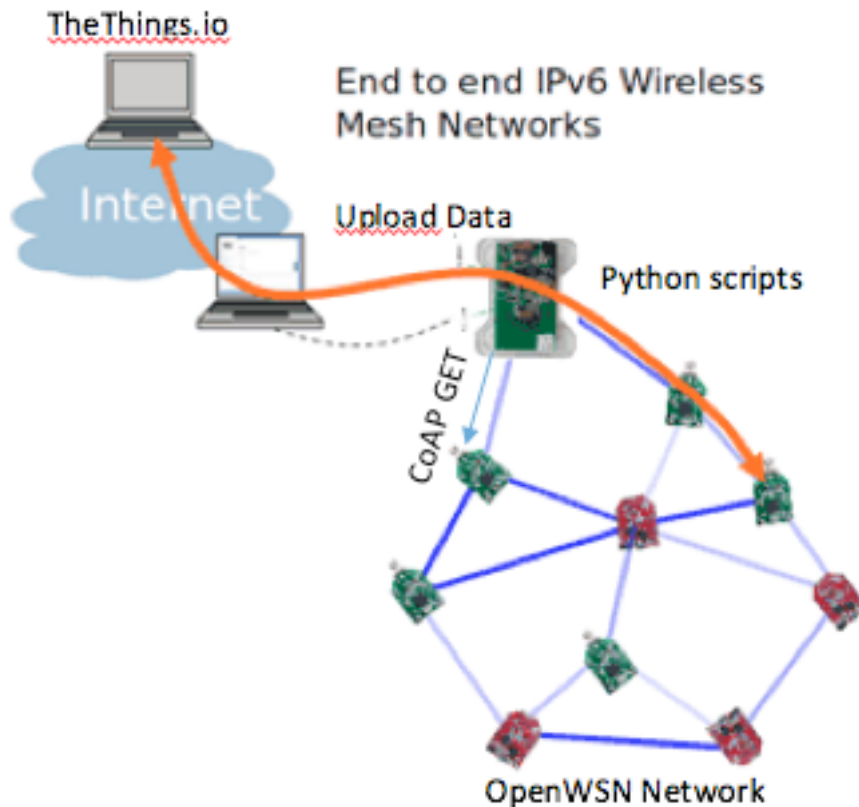


Figura 31. Arquitectura del sistema[29].

3.2 Configuración de software.

La configuración de software se puede dividir en distintas partes, por un lado el bloque sensor donde están situados los OpenMote-cc2538 que llevan el firmware en lenguaje C, un bloque de procesamiento donde mediante scripts en Python se realizan solicitudes GET a los mote y se procesan los datos que se quieren enviar al cloud y se almacenan en una base de datos, el último punto será la plataforma Thethings.io donde enviaremos los datos procesados para su tratamiento.

3.2.1 Instalación OpenWSN.

La instalación se puede realizar bajo distintos sistemas operativos, Windows, Linux y Mac OSX, en nuestro caso se hace bajo una versión de Debian para Raspberry pi (Jessie[34]). Como vamos a realizar la instalación desde los propios repositorios de github [35], primero hemos de instalar la herramienta “git”:

1. `sudo apt-get update`
2. `sudo apt-get install git`

Una vez instalado, creamos una carpeta donde se vaya a realizar a instalación que será desde donde realizaremos la descarga de los ficheros, en este caso instalaremos openwsn-fw, openwsn-sw y CoAP.

1. /openwsn/sudo git clone https://github.com/openwsn-berkeley/openwsn-fw.git
2. /openwsn/sudo git clone https://github.com/openwsn-berkeley/openwsn-sw.git
3. /openwsn/sudo git clone https://github.com/openwsn-berkeley/coap.git

Una vez instalado comprobamos que estamos utilizando la última versión, se puede realizar la comprobación y update cuando se quiera utilizando los siguientes comandos:

1. openwsn\$ cd openwsn-sw/
2. openwsn/openwsn-sw\$ git pull
3. Already up-to-date.
4. openwsn/openwsn-sw\$ cd ..
5. openwsn\$ cd openwsn-fw/
6. openwsn/openwsn-fw\$ git pull
7. Already up-to-date.

Ahora se tienen que instalar las siguientes herramientas para poder compilar los firmware, el compilador gcc, scons, Python:

1. sudo apt-get install python-dev
2. sudo apt-get install scons
3. sudo apt-get install gcc*version*

Otra herramienta útil es el plugin para Firefox Copper [36], es un agente que nos permite manejar el esquema CoAP para las URI para poder ver los mote e interactuar con los elementos de IoT una vez que está arrancada la simulación en OVS.

Una vez realizada la compilación de los mote ejecutamos OpenVisualizer para ver la interacción de la simulación, en este ejemplo se indica una topología lineal, aunque puede ser mallada y también se puede indicar el número de motes a simular.

1. /openwsn/openwsn-sw/software/openvisualizer# sudo scons run web --sim --simTopology=Linear

```
rubo@rubo:~/Escritorio/openwsn/openwsn-sw/software/openvisualizer$ sudo scons runweb --sim --simTopology=linear
[sudo] password for rubo:
scons: Reading SConscript files ...

┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐
│   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │
│   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │
└───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘
openwsn.org

scons: done reading SConscript files.
scons: Building targets ...
Copy("bin/openVisualizerApp/sim_files", "../../openwsn-fw/bsp/boards/python/openwsnmodule_obj.h")
Mkdir("bin/openVisualizerApp/sim_files/linux")
Copy("bin/openVisualizerApp/sim_files/linux/oos_openwsn-amd64.so", "../../openwsn-fw/build/python/gcc/projects/common/oos_openwsn.so")
Copy("bin/openVisualizerApp/sim_files", "../../openwsn-fw/build/python/gcc/projects/common/oos_openwsn.so")
Delete("build/runui/web_files")
Mkdir("/home/rubo/Escritorio/openwsn/openwsn-sw/software/openvisualizer/build/runui")
Copy("build/runui/web_files", "bin/openVisualizerApp/web_files")
Delete("build/runui/sim_files")
Mkdir("/home/rubo/Escritorio/openwsn/openwsn-sw/software/openvisualizer/build/runui")
Copy("build/runui/sim_files", "bin/openVisualizerApp/sim_files")
uiRunner(["bin/openVisualizerApp/openVisualizerWeb"], ["bin/openVisualizerApp/openVisualizerWeb.py"])
ioctl(TUNSETIFF): Device or resource busy

created following virtual interface:
18: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet6 bbbb::1/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::1/64 scope link
        valid_lft forever preferred_lft forever
12:13:07 INFO create instance
12:13:07 INFO create instance
12:13:07 INFO create instance
```

Figura 34. Ejecución OVS.

Una vez iniciado desde un navegador web accedemos por https a nuestra loopback para abrir OVS:

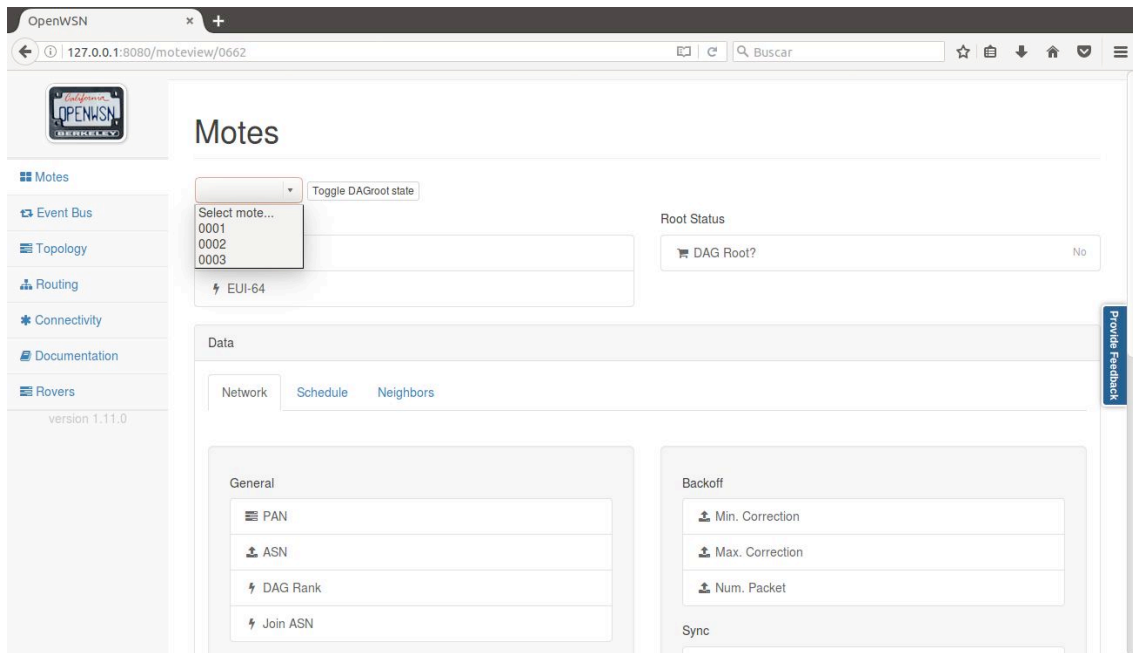


Figura 35. Motes de una simulación.

Para poder crear la red hay que declarar un mote como el DAGRoot, en esta función hace de bridge entre los mote y OpenWSN, debe estar conectado al sistema donde está Openwsn.

Motes

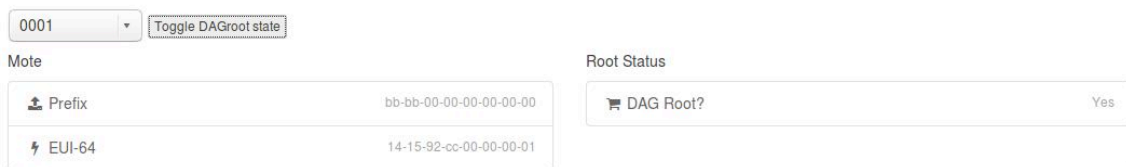


Figura 36. DAGRoot.

En la pestaña neighbours podemos ver los mote que han sincronizado y que son vecinos del DAGroot, podemos ver los siguientes campos, la dirección MAC, DAG Rank, cuando más bajo más posibilidades de ser el DAGroot, el nivel RSS, etc.

Used	Insecure	f6PNORES	sixtopGEN	sixtopSeqNum	Parent	Stable	Stability	Address	DAG Rank	JP	RSS	RX	TX	TX ACK
1	1	0	0	0	1	1	0	14-15-92-cc-00-00-00-01 (64b)	256	0	-50 dBm	159	10	9
1	1	0	0	0	0	1	0	14-15-92-cc-00-00-00-03 (64b)	5376	20	-50 dBm	110	0	0

Figura 37. Vecinos DAGRoot.

Desde el navegador también podemos ver la topología, en nuestro caso lineal (sólo se puede ver en una simulación, al compilar sobre hardware no funciona):



Figura 38. Topología.

En la pestaña routing podemos ver como el RPL forma el routing:

Routing

Current RPL Routing

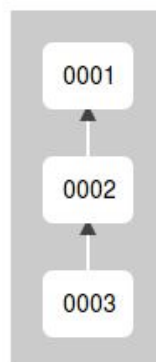


Figura 39. Routing.

Desde el terminal donde ejecutamos OVS podemos ver como se reciben mediante RPL paquetes Destination Advertisement Object (DAO) desde los mote, con su IPv6, la de su nodo padre y los hijos. La ipv6 se forma empezando siempre por el prefijo bbbb: seguido de la dirección MAC del mote, en el caso de ser virtual sería bbbb:0:0:0:1415:92cc:0:X siendo X valores en aumento a partir del 1.

```
received RPL DAO from bbbb:0:0:0:1415:92cc:0:2
- parents:
  bbbb:0:0:0:1415:92cc:0:1
- children:
  bbbb:0:0:0:1415:92cc:0:3

received RPL DAO from bbbb:0:0:0:1415:92cc:0:3
- parents:
  bbbb:0:0:0:1415:92cc:0:2
- children:
```

Figura 40. RPL DAO

Podemos comprobar cómo responden a ping:

```
rubo@rubo:~$ ping6 bbbb:0:0:0:1415:92cc:0:3
PING bbbb:0:0:0:1415:92cc:0:3(bbbb::1415:92cc:0:3) 56 data bytes
64 bytes from bbbb::1415:92cc:0:3: icmp_seq=1 ttl=64 time=224 ms
64 bytes from bbbb::1415:92cc:0:3: icmp_seq=2 ttl=64 time=221 ms
64 bytes from bbbb::1415:92cc:0:3: icmp_seq=3 ttl=64 time=220 ms
^C
--- bbbb:0:0:0:1415:92cc:0:3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 220.173/221.762/224.004/1.718 ms
rubo@rubo:~$ ping6 bbbb:0:0:0:1415:92cc:0:2
PING bbbb:0:0:0:1415:92cc:0:2(bbbb::1415:92cc:0:2) 56 data bytes
64 bytes from bbbb::1415:92cc:0:2: icmp_seq=1 ttl=64 time=115 ms
64 bytes from bbbb::1415:92cc:0:2: icmp_seq=2 ttl=64 time=131 ms
^C
--- bbbb:0:0:0:1415:92cc:0:2 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2000ms
rtt min/avg/max/mdev = 115.106/123.338/131.570/8.232 ms
```

Figura 41. Icmp IPv6.

3.3 Hardware

El hardware utilizado consta de una Raspberry pi, las placas OpenMote y OpenUsb.

- Raspberry pi: se trata de un ordenador de tamaño reducido (SBC) de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi[37], con el propósito de estimular la enseñanza de ciencias de computación en las escuelas.

El modelo utilizado es el 3B con las siguientes especificaciones:

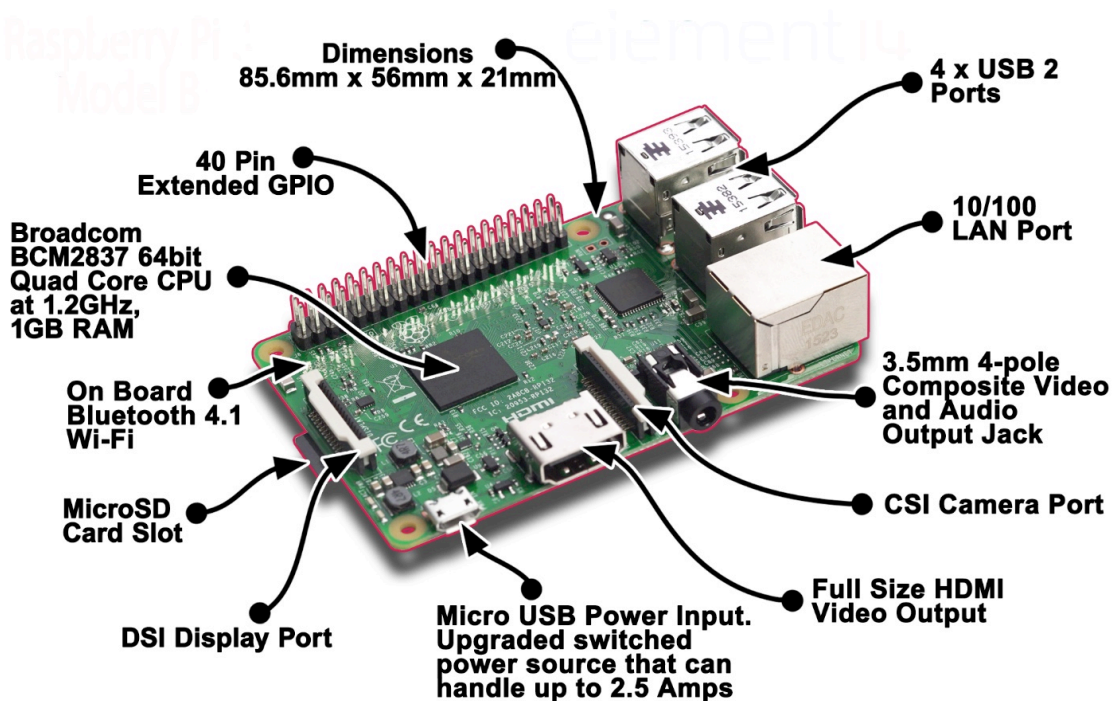


Figura 42. Raspberry Pi 3[33].

- OpenMote: tiene un chip CC2538 fabricado por Texas instrument [38], se trata de un SoC con microcontrolador ARM Cortex-M3, 512 Kb de memoria flash programable, 32Kb de memoria RAM y RF de 2,4Ghz IEEE 802.15.4.

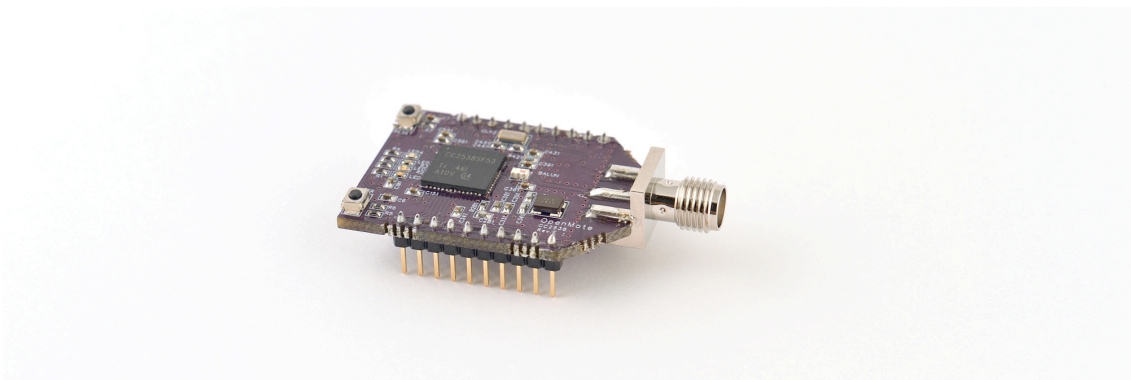


Figura 43. OpenMote[34].

- OpenUsb: esta placa permite la conexión por puerto usb del OpenMote para realizar la compilación o dejarlo como DAGroot, cuenta con espacio para dos pilas AA para alimentación independiente, además incorpora un sensor de humedad y temperatura STH21, un sensor de luminosidad MAX44009 y un sensor de aceleración ADXL346.



Figura 44. OpenUsb con OpenMote[35].

3.4 Compilación sobre OpenMote.

Para poder compilar sobre las placas OpenMote primero debemos descargar el toolchain necesario ya que el chip es un ARM M3, se puede descargar desde este enlace: <https://launchpad.net/gcc-arm-embedded>, una vez descargado hay que descomprimirlo dentro de la carpeta /openwsn-fw/bootloader/

La compilación se realiza de la misma manera que para la simulación sólo que se indica que va a ser sobre una plataforma hardware, el puerto USB donde está conectado el mote usb y también le indicamos la aplicación a compilar.

Antes de poder compilar hay que indicar en determinados ficheros que inicialice la aplicación que vamos a introducir y la ruta donde se encuentra, los ficheros a modificar son:

- Opendefs.h: se realiza un registro de cada aplicación con un ID único dentro de OpenWSN que indica quién genera los datos.


```

161 // applications
162 COMPONENT_C6T = 0x1b,
163 COMPONENT_CEXAMPLE = 0x1c,
164 COMPONENT_CINFO = 0x1d,
165 COMPONENT_CLEDS = 0x1e,
166 COMPONENT_CSENSORS = 0x1f,
167 COMPONENT_CSTORM = 0x20,
168 COMPONENT_CWELLKNOWN = 0x21,
169 COMPONENT_TECHO = 0x22,
170 COMPONENT_TOHLONE = 0x23,
171 COMPONENT_UECHO = 0x24,
172 COMPONENT_UINJECT = 0x25,
173 COMPONENT_RRT = 0x26,
174 COMPONENT_SECURITY = 0x27,

```

Figura 45. Opendesf.h

- Sconscript.env: indica el path a los ficheros fuente de cada aplicación para poder realizar la compilación.

```

71 os.path.join('#', 'build', 'python_gcc', 'openstack', '02b-MACHigh'),
72 os.path.join('#', 'build', 'python_gcc', 'openstack', '03a-IPHC'),
73 os.path.join('#', 'build', 'python_gcc', 'openstack', '03b-IPv6'),
74 os.path.join('#', 'build', 'python_gcc', 'openstack', '04-TRAN'),
75 os.path.join('#', 'build', 'python_gcc', 'openstack', 'cross-layers'),
76 # openapps
77 os.path.join('#', 'build', 'python_gcc', 'openapps'),
78 os.path.join('#', 'build', 'python_gcc', 'openapps', 'opencoap'),
79 os.path.join('#', 'build', 'python_gcc', 'openapps', 'c6t'),
80 os.path.join('#', 'build', 'python_gcc', 'openapps', 'rrt'),
81 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cexample'),
82 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cinfo'),
83 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cleds'),
84 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cstorm'),
85 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cwellknown'),
86 os.path.join('#', 'build', 'python_gcc', 'openapps', 'techo'),
87 os.path.join('#', 'build', 'python_gcc', 'openapps', 'tohlone'),
88 os.path.join('#', 'build', 'python_gcc', 'openapps', 'uecho'),
89 os.path.join('#', 'build', 'python_gcc', 'openapps', 'uexpiration'),
90 os.path.join('#', 'build', 'python_gcc', 'openapps', 'uexpiration_monitor'),
91 os.path.join('#', 'build', 'python_gcc', 'openapps', 'uinject'),
92 os.path.join('#', 'build', 'python_gcc', 'openapps', 'userialbridge'),
93 os.path.join('#', 'build', 'python_gcc', 'openapps', 'cjoin'),
94 os.path.join('#', 'build', 'python_gcc', 'openapps', 'csensors'),
95 ]

```

Figura 46. Sconscript.env.

- Sconscript: aquí se almacenan las rutas a las carpetas donde están las aplicaciones, las nuevas aplicaciones se tienen que referenciar en este archivo.

```

102     # openapps
103     # TODO: remove once cleaned-up?
104     os.path.join('#', 'openapps', 'opencoap'),
105     os.path.join('#', 'openapps', 'c6t'),
106     os.path.join('#', 'openapps', 'cexample'),
107     os.path.join('#', 'openapps', 'cinfo'),
108     os.path.join('#', 'openapps', 'cleds'),
109     os.path.join('#', 'openapps', 'cstorm'),
110     os.path.join('#', 'openapps', 'cwellknown'),
111     os.path.join('#', 'openapps', 'rrt'),
112     os.path.join('#', 'openapps', 'techo'),
113     os.path.join('#', 'openapps', 'tohlone'),
114     os.path.join('#', 'openapps', 'uecho'),
115     os.path.join('#', 'openapps', 'uinject'),
116     os.path.join('#', 'openapps', 'userialbridge'),
117     os.path.join('#', 'openapps', 'cjoin'),
118     os.path.join('#', 'openapps', 'uexpiration'),
119     os.path.join('#', 'openapps', 'uexpiration_monitor'),
120     os.path.join('#', 'openapps', 'csensors'),

```

Figura 47. Sconscript.

- Openapps.c: en el compilador se tiene que indicar las aplicaciones a añadir, los ficheros .h y la inicialización de la nueva app.

```


10 #include "opencoap.h"
11 #include "c6t.h"
12 #include "cinfo.h"
13 #include "cleds.h"
14 #include "cjoin.h"
15 #include "cwellknown.h"
16 #include "rrt.h"
17 #include "csensors.h"
18 // UDP
19 #include "uecho.h"
20 #include "uinject.h"
21 #include "userialbridge.h"
22 #include "uexpiration.h"
23 #include "uexpiration_monitor.h"
24
25 //===== variables =====
26
27 //===== prototypes =====
28
29 //===== public =====
30
31 //===== private =====
32
33 void openapps_init(void) {
34     //-- 04-TRAN
35     opencoap_init();    // initialize before any of the CoAP applications
36
37     // CoAP
38     //c6t_init();
39     cinfo_init();
40     cleds_init();
41     //cjoin_init();
42     cwellknown_init();
43     csensors_init();

```

Figura 48. Openapps.c.

Después de la modificación de los ficheros anteriores ya podemos realizar la compilación para la aplicación “csensors”.

```
rubo@rubo:~/Escritorio/openwsn/openwsn-fw$ sudo scons board=openmote-cc2538 toolchain=armgcc app=csensors bootload=/dev/ttyUSB0 oos_op
enwsn
scons: Reading SConscript files ...



none
scons: done reading SConscript files.
scons: Building targets ...
Dynifying build/openmote-cc2538_armgcc/openapps/openapps_dyn.c
arm-none-eabi-size --format=berkeley -x --totals build/openmote-cc2538_armgcc/projects/common/03oos_openwsn_prog
  text  data  bss  dec  hex filename
0x196fc 0x400 0x1e00 112892 1b8fc build/openmote-cc2538_armgcc/projects/common/03oos_openwsn_prog
0x196fc 0x400 0x1e00 112892 1b8fc (TOTALS)
OpenMoteCC2538_bootload(["build/openmote-cc2538_armgcc/projects/common/03oos_openwsn_prog.phonyupload"], ["build/openmote-cc2538_armgc
c/projects/common/03oos_openwsn_prog.ihex"])
starting bootloading on /dev/ttyUSB0
Opening port /dev/ttyUSB0, baud 400000
Reading data from build/openmote-cc2538_armgcc/projects/common/03oos_openwsn_prog.ihex
Your firmware looks like an Intel Hex file
Connecting to target...
CC2538 PG2.0: 512KB Flash, 32KB SRAM, CCFG at 0x0027FFD4
Primary IEEE Address: 06:13:0A:8E:00:12:4B:00
Erasing 524288 bytes starting at address 0x00200000
  Erase done
Writing 524256 bytes starting at address 0x00200000
Write 232 bytes at 0x0027FEF80
  Write done
done bootloading on /dev/ttyUSB0
scons: done building targets.
```

Figura 49. Compilación OpenMote.

Podemos comprobar rápidamente que nos devuelve la información solicitada mediante Copper, utilizando en la URI el path a la aplicación que hemos compilado:

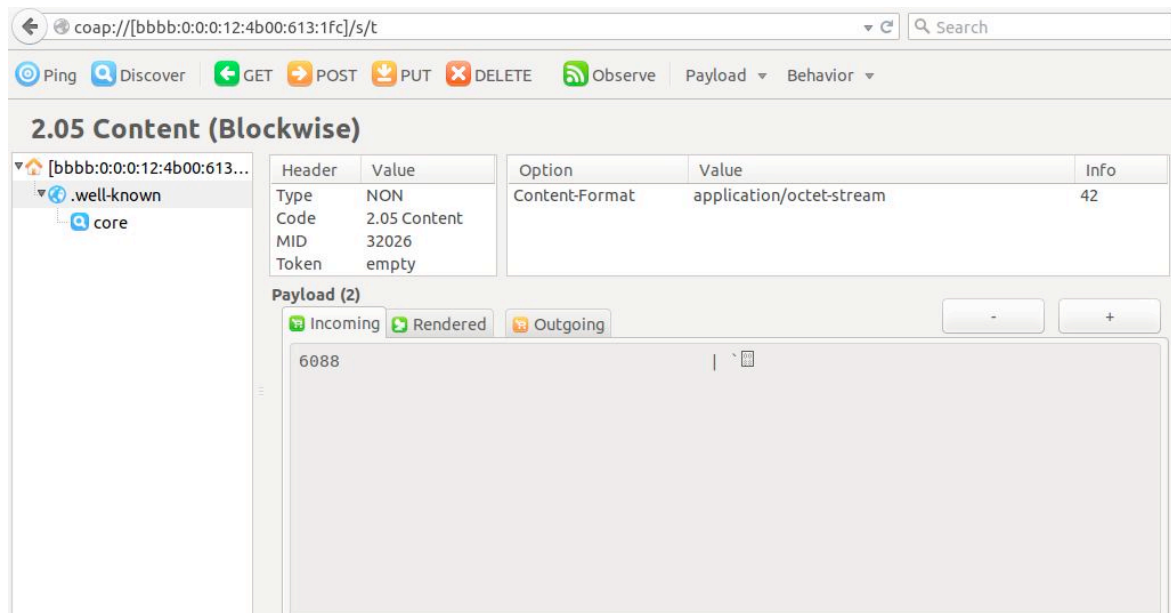


Figura 50. Captura de datos mediante Copper.

3.5 Instalación BBDD en Raspberry Pi.

Para almacenar los datos en la Raspberry Pi se utiliza como base de datos MariaDB para Debian Jessie, una base de datos derivado de MySQL diseñado por Michael Widenius [40] fundador de MySQL. Para la instalación debemos ejecutar:

```
1. sudo apt-get install mariadb-server mariadb-client
```

Ahora escribiendo `mysql` entraríamos en nuestra bbdd y nos aporta la información de la versión instalada.

```
[root@raspberrypi:~# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 10.1.23-MariaDB-9+deb9u1 Raspbian 9.0

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Figura 51. Versión MariaDB.

3.6 Descripción del firmware.

En este apartado se realiza una descripción del firmware desarrollado, desde los scripts de Python, la base de datos para guardar las mediciones en la Raspberry pi a la conexión con la plataforma Thethings.io.

3.6.1 Scripts Python.

Los scripts de Python son los encargados de generar los datos aleatorios de los sensores simulando la captura de datos en OpenMote, convertir los valores de las mediciones, enviar los datos a una base de datos para tenerlos almacenados y enviar los datos ya procesados a la plataforma Cloud.

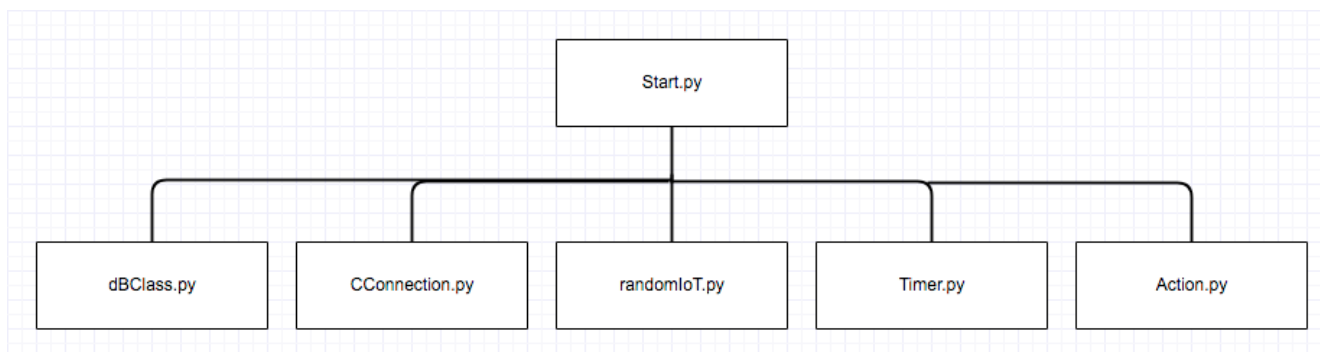


Figura 52. Esquema Scripts Python.

- Start.py: la ejecución de este script pone en funcionamiento el sistema entero, realiza las llamadas necesarias a las distintas clases.

```

1 import sys
2 from time import sleep
3 from randomIot import RandomIot
4 from timer import RepeatedTimer
5 from dbClass import Database
6 from actionTrigger import action
7 from cloudConnector import thethings
8
9 def setTempInt(random, db, thethings):
10     data = action().lowTemp(random.getTempInt,21)
11     db.insertData(db,data,'internal','temp')
12     thethings.sendData('temperatureInt',data)
13
14 def setTempExt(random, db, thethings):
15     data = action().lowTemp(random.getTempExt,19)
16     db.insertData(db,data,'external','temp')
17     thethings.sendData('temperatureExt',data)
18
19 def setHumInt(random, db, thethings):
20     data = str(random.getHumInt())
21     db.insertData(db,data,'internal','hum')
22     thethings.sendData('humidityInt',data)
23
24 def setHumExt(random, db, thethings):
25     data = str(random.getHumExt())
26     db.insertData(db,data,'external','hum')
27     thethings.sendData('humidityExt',data)
28
29 db = Database()
30 random = RandomIot()
31 thethings = thethings("19sbFlk2ZtBRc7fNwagHZ-MgyrgQjaZP1gI9UbS5Aso")
32 print "starting.."
33 rt = RepeatedTimer(60, setTempInt, setTempExt, setHumInt, setHumExt, random, db, thethings)

```

Figura 53. Start.py.

Se definen 4 funciones para la ejecución de las distintas tareas:

Def setTempInt, setTempExt, setHumint y setHumExt, cada una de ellas para la simulación de cada sensor de temperatura y humedad tanto interno como externo.

Después se realiza la inicialización de los objetos de las diferentes clases que hacen posible la integración del sistema. Primero se crea un objeto para el control de la base de datos, otro para la generación de datos aleatorios y el último para obtener una conexión con la plataforma Cloud utilizando nuestro token (valor de conector con el Cloud) asignado a nuestra plataforma IoT.

Se pasa a la ejecución de las funciones de forma multithreading utilizando para ello la clase RepeatedTimer. Cada función primeramente realiza una llamada a un actuador el cuál ejecuta mediante un callback el método de generación aleatorio y analiza este dato para poder dotar de una funcionalidad de actuación o función dependiendo del valor recibido antes de pasarlo para su análisis.

Finalmente se guarda en una base de datos el valor aleatorio generado en función de si es temperatura o humedad y la localización externo o interno y a su vez se envían los datos al Cloud.

- dBclass.py: realiza la conexión con nuestra base de datos local en la Raspberry pi, realiza la inserción de los datos aleatorios generados añadiendo la fecha y hora a cada medida.

Tiene un constructor el cuál inicializa la conexión con nuestra base de datos y el conjunto de métodos para la manipulación de la información, consta de un magic method para la desconexión de la base de datos cuando se finalice la ejecución del script start.py.

```

1 import MySQLdb
2 import datetime
3 class Database:
4     host = 'localhost'
5     user = 'root'
6     password = 'password'
7     db = 'IOTDB'
8
9     def __init__(self):
10        self.connection = MySQLdb.connect(self.host, self.user, self.password, self.db)
11        self.cursor = self.connection.cursor()
12
13    def insert(self, query):
14        try:
15            self.cursor.execute(query)
16            self.connection.commit()
17            return True
18        except:
19            self.connection.rollback()
20            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
21            message = template.format(type(ex).__name__, ex.args)
22            print message
23
24    def query(self, query):
25        cursor = self.connection.cursor( MySQLdb.cursors.DictCursor )
26        cursor.execute(query)
27        return cursor.fetchall()
28
29    def __del__(self):
30        self.connection.close()
31
32    def insertData(self,db,value,name,type):
33        query = "INSERT INTO IOTDB.data (value, name, timestamp, type) VALUES (" + "'" + value + "', " + "'" + name + "', " + "'" + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "', " + type + ")"
34        if self.insert(query):
35            print 'STORED: ' + value

```

Figura 54. dBclass.py.

- RandomIoT.py: realiza la generación aleatoria de información de los distintos sensores simulados.

Consta de 4 métodos que involucran las medidas de temperatura interior y exterior, así como las de humedad.

Para dotar a la temperatura exterior de mayor realismo se ha optado por generar un método que modifique el valor generado de temperatura dependiendo de la hora local donde se ejecuta el sistema.

Los valores son devueltos como un entero y son modificables dentro del propio código.

```
1 import random
2 import decimal
3 import datetime as dt
4
5 class RandomIot:
6
7     def __init__(self):
8         self.data = None
9
10    def getHourRangeTempExt(self):
11        hour = dt.datetime.now().hour
12        self.HourTempMax = (int(hour) + 1) * 100
13        self.HourTempMin = (int(hour) - 1) * 100
14
15    def getTempInt(self):
16        return float(decimal.Decimal(random.randrange(2000, 2100))/100)
17
18    def getTempExt(self):
19        self.getHourRangeTempExt()
20        return float(decimal.Decimal(random.randrange(self.HourTempMin, self.HourTempMax))/100)
21
22    def getHumInt(self):
23        return float(decimal.Decimal(random.randrange(5000, 5200))/100)
24
25    def getHumExt(self):
26        return float(decimal.Decimal(random.randrange(6300, 6500))/100)
```

Figura 55. RandomIoT.py.

- Timer.py: dota de un sistema multihilo para que las 4 funciones de generación aleatoria se ejecuten de forma simultánea, implementando un método de parada en caso de ser necesario.

Esta clase es personalizable, es decir puede ejecutar cualquier función con cualquier tipo de argumentos.

```

1 from threading import Timer
2
3 class RepeatedTimer(object):
4     def __init__(self, interval, functionA, functionB, functionC, functionD, *args, **kwargs):
5         self._timer = None
6         self.interval = interval
7         self.functionA = functionA
8         self.functionB = functionB
9         self.functionC = functionC
10        self.functionD = functionD
11        self.args = args
12        self.kwargs = kwargs
13        self.is_running = False
14        self.start()
15
16    def _run(self):
17        self.is_running = False
18        self.start()
19        self.functionA(*self.args, **self.kwargs)
20        self.functionB(*self.args, **self.kwargs)
21        self.functionC(*self.args, **self.kwargs)
22        self.functionD(*self.args, **self.kwargs)
23
24    def start(self):
25        if not self.is_running:
26            self._timer = Timer(self.interval, self._run)
27            self._timer.start()
28            self.is_running = True
29
30    def stop(self):
31        self._timer.cancel()
32        self.is_running = False

```

Figura 56. Timer.py.

- CConnector: en su constructor se realiza la conexión con la API de Thethings.io mediante el token propio asignado a nuestro elemento IoT. El método sendData conecta mediante TheThingsAPI para realizar el envío de la información generada aleatoriamente al Cloud utilizando el token asignado en TheThings.io a nuestro elemento IoT, esta transacción está implementada con un sistema de excepción para alertar en caso de error.

```

1 from thethings import ThethingsAPI
2
3 class thethings:
4
5     def __init__(self, token):
6         self.connector = ThethingsAPI(token)
7
8     def sendData(self, name, value):
9         try:
10            self.connector.addVar(name, value)
11            self.connector.write()
12            print 'SENT DATA TO THINGS.IO'
13        except Exception as ex:
14            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
15            message = template.format(type(ex).__name__, ex.args)
16            print message

```


Figura 57. CConnection.py.

- ActionTrigger: ejecuta el método de generación aleatoria mediante callback para analizar la información generada antes de ser enviada y guardada en la base de datos. Este trigger es configurable dependiendo del valor obtenido para ejecutar distintas funciones, ya sea mandar una señal de activación para encender la calefacción o alertar de cualquier eventualidad en el dato, este script es propio solo para la simulación de fog computing donde la toma de decisiones y actuaciones se realizan en el elemento edge.

```
1 class action:
2
3 def lowTemp(self, callback, minimum):
4     data = callback()
5     if data < minimum:
6         '''
7         IMPLEMENT ACTIONS TO INTERACT WITH OTHER DEVICES/TRIGGER CLOUD/GPIO
8         '''
9     print "WARNING "+ str(callback.__name__) + "
10 : Minimum temp defined is: " + str(minimum) + ", actual: " + str(data)
11     return str(data)
```

Figura 58. ActionTrigger.py.

Sí se desea que el script start.py esté residente en memoria en segundo plano ejecutándose de manera indefinida y al mismo tiempo analizar sus entradas y salidas en tiempo real se ejecutaría de la siguiente manera:

1. #nohup python start.py &

```
[root@raspberrypi:/home/pi/Desktop/Scripts/thethingsIoT# cat nohup.out
starting...
WARNING getTempInt: Minimum temp defined is: 21, actual: 20.28
STORED: 20.28
SENT DATA TO THINGS.IO
WARNING getTempExt: Minimum temp defined is: 19, actual: 18.75
STORED: 18.75
SENT DATA TO THINGS.IO
STORED: 51.38
SENT DATA TO THINGS.IO
STORED: 63.74
SENT DATA TO THINGS.IO
WARNING getTempInt: Minimum temp defined is: 21, actual: 20.74
STORED: 20.74
SENT DATA TO THINGS.IO
WARNING getTempExt: Minimum temp defined is: 19, actual: 18.15
STORED: 18.15
SENT DATA TO THINGS.IO
STORED: 50.02
SENT DATA TO THINGS.IO
STORED: 64.29
SENT DATA TO THINGS.IO
WARNING getTempInt: Minimum temp defined is: 21, actual: 20.68
STORED: 20.68
SENT DATA TO THINGS.IO
WARNING getTempExt: Minimum temp defined is: 19, actual: 17.21
STORED: 17.21
SENT DATA TO THINGS.IO
STORED: 51.22
SENT DATA TO THINGS.IO
STORED: 63.08
SENT DATA TO THINGS.IO
```

Figura 59. nohup.

En la captura se puede observar el valor aleatorio generado, el mensaje de que se guarda en la base de datos y también se envía al Cloud, pero al estar actuando el Trigger también indica el mensaje de Warning.

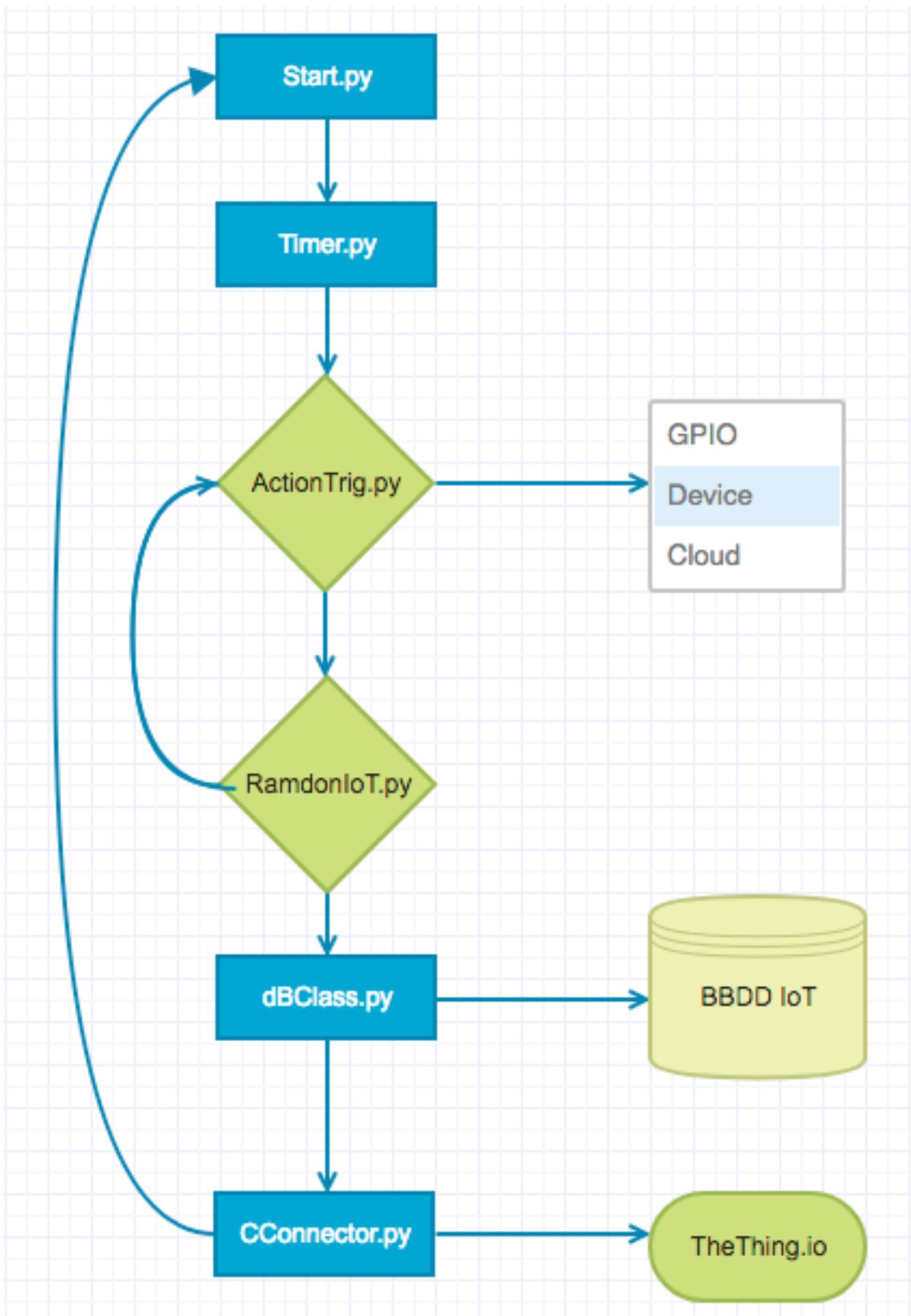


Figura 60. Ejecución Scripts.

3.6.2 Base de datos.

Como se comentó anteriormente los datos generados de manera aleatoria se almacenarán en una base de tipo sql dentro de la Raspberry pi.

Una vez que tenemos el software instalado creamos nuestra base de datos para almacenar la información generada por los scripts con los siguientes campos:

- Id: número correspondiente a la medida generada aleatoriamente.
- Value: valor de la medida generada por los scripts, ya sea de temperatura o humedad.
- Name: internal o external, dependiendo de la localización del sensor.
- TimeStamp: fecha y hora de la generación del dato.
- Type: define si es medida de temperatura o humedad.

```
[MariaDB [(none)]> show columns from data from IOTDB;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |
| value      | varchar(45)   | YES  |     | NULL    |                |
| name       | varchar(45)   | YES  |     | NULL    |                |
| timestamp  | varchar(45)   | YES  |     | NULL    |                |
| type       | varchar(45)   | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

Figura 61. IoTDB.

Cuando se realiza la ejecución de RandomIoT.py el dato es agregado a la tabla de la siguiente manera:

```

[MariaDB [(none)]]> USE IOTDB;
Reading table information for completion of table and column name
You can turn off this feature to get a quicker startup with -A

Database changed
[MariaDB [IOTDB]]> select * from data;
+-----+-----+-----+-----+-----+
| id    | value | name      | timestamp                               | type |
+-----+-----+-----+-----+-----+
| 1     | 20.08 | internal  | 2017-12-06 18:36:43.660833            | temp |
| 2     | 18.62 | external  | 2017-12-06 18:36:43.928529            | temp |
| 3     | 51.91 | internal  | 2017-12-06 18:36:44.125200            | hum   |
| 4     | 63.98 | external  | 2017-12-06 18:36:44.314414            | hum   |
| 5     | 20.41 | internal  | 2017-12-06 18:37:43.662053            | temp |
| 6     | 17.7  | external  | 2017-12-06 18:37:43.870064            | temp |
| 7     | 50.68 | internal  | 2017-12-06 18:37:44.074926            | hum   |
| 8     | 64.46 | external  | 2017-12-06 18:37:44.278225            | hum   |
| 9     | 20.43 | internal  | 2017-12-06 18:38:43.663212            | temp |
| 10    | 17.24 | external  | 2017-12-06 18:38:43.910823            | temp |
| 11    | 51.75 | internal  | 2017-12-06 18:38:44.161455            | hum   |
| 12    | 63.76 | external  | 2017-12-06 18:38:44.368459            | hum   |
| 13    | 20.04 | internal  | 2017-12-06 18:39:43.663987            | temp |
| 14    | 18.02 | external  | 2017-12-06 18:39:43.842819            | temp |
| 15    | 51.6  | internal  | 2017-12-06 18:39:44.028518            | hum   |
| 16    | 63.84 | external  | 2017-12-06 18:39:44.327659            | hum   |
| 17    | 20.16 | internal  | 2017-12-06 18:40:43.665232            | temp |
| 18    | 17.61 | external  | 2017-12-06 18:40:43.846832            | temp |
| 19    | 50.31 | internal  | 2017-12-06 18:40:44.032821            | hum   |
| 20    | 63.23 | external  | 2017-12-06 18:40:44.322213            | hum   |
| 21    | 20.36 | internal  | 2017-12-06 18:41:43.666439            | temp |
| 22    | 17.66 | external  | 2017-12-06 18:41:43.894223            | temp |
| 23    | 50.83 | internal  | 2017-12-06 18:41:44.143848            | hum   |
| 24    | 63.43 | external  | 2017-12-06 18:41:44.336589            | hum   |
| 25    | 20.5  | internal  | 2017-12-06 18:42:43.667658            | temp |
| 26    | 18.59 | external  | 2017-12-06 18:42:43.943222            | temp |
| 27    | 51.74 | internal  | 2017-12-06 18:42:44.207964            | hum   |
| 28    | 64.51 | external  | 2017-12-06 18:42:44.415141            | hum   |
| 29    | 20.13 | internal  | 2017-12-06 18:43:43.668425            | temp |
| 30    | 18.6  | external  | 2017-12-06 18:43:43.950822            | temp |
| 31    | 50.64 | internal  | 2017-12-06 18:43:44.185228            | hum   |
| 32    | 64.07 | external  | 2017-12-06 18:43:44.400335            | hum   |
| 33    | 20.86 | internal  | 2017-12-06 18:44:43.669641            | temp |
| 34    | 18.5  | external  | 2017-12-06 18:44:43.863578            | temp |

```

Figura 62. Agregación de datos.

En la figura podemos observar el id con el que se agrega dentro de la BBDD, el valor de la medición generada, si es tipo interno o externo, la fecha y hora, así como el tipo de sensor que es (humedad o temperatura).

3.7 Subida de datos a la nube.

Para poder realizar la subida de datos a la nube una vez tenemos la cuenta creada en Thethings.io el primer paso a realizar es la creación del elemento que va enviar la información al Cloud denominado thing:

The screenshot shows a web form titled "Create a new IoT product". It has the following elements:

- Name ***: A text input field with the placeholder "Here goes your awesome IoT product name".
- Board ***: A dropdown menu with a checkmark icon. The list includes: Akeru/Smart Everything, Arduino, **Beaglebone** (highlighted), Electric IMP, ESP8266, Intel Edison, Raspberry Pi, Zolertia, and Other.
- Do you need help choosing the right device?**: A link to "see how to choose the right device".
- MessagePack** and **Protocol Buffer**: Two radio button options.
- Opendata**: A checkbox with the text "The Open Data option gives public access to your data allowing other users to read it".
- Buttons**: "CANCEL" and "CREATE" buttons at the bottom.

Figura 63. Creación del thing.

Una vez que tenemos creado nuestro equipo, podemos asignarle un token único que identificará los datos que enviamos al Cloud con nuestro thing dentro de Activate more tokens nos aparecen códigos de activación, una vez seleccionado el que deseemos hay que pulsar sobre Activate.

Your thing token:

```
mu_ihCVpWjSL-NBAkIYhnCEKbzBakjXTTqqO09j3-zM
```

Copy and paste this code to your device and start sending data to thethings.io:

javascript

python

```
#!/usr/bin/python

from theThingsAPI import TheThingsAPI

tt = TheThingsAPI("YourThingToken")

# Read example
response = tt.read("yourVariable")
print 'value: ' + response[0]['value'] + ' date: ' + response[0]
['datetime']

# write example
tt.addVar('YourVariable_1', 'YourValue_1')
tt.addVar('YourVariable_2', 'YourValue_2')
tt.write();
```

Download the libraries from [GitHub](#) or check the [developers page](#) to create your own through thethings.io API.

Figura 64. Código envío datos con token.

En nuestro sistema esta llamada al API de Thethings.io con nuestro token lo realizamos con el script Python CConnector.py, una vez la Raspberry pi está enviando datos podemos crear una serie de dashboards para visualizar la información de una forma sencilla, elegimos el nombre de la gráfica, el origen de los datos, en nuestro caso del thing asociado al token y el origen que serán las variables de datos que estamos enviando.

 Basic Configuration

Widget Name	<input type="text" value="Example"/>
Data Source	<input type="text" value="Thing Resource"/>
Product	<input type="text" value="Rasp"/>
Thing	<input type="text" value="Ras"/>
Resource	<ul style="list-style-type: none">✓ctm_payloaddescriptionhumidityhumidityExthumidityInttemperaturetemperatureExttemperatureInt

Figura 65. Creación Dashboard.

El resultado final lo podemos observar en la siguiente figura donde se han representado los datos de los distintos sensores simulados durante un periodo de tiempo.

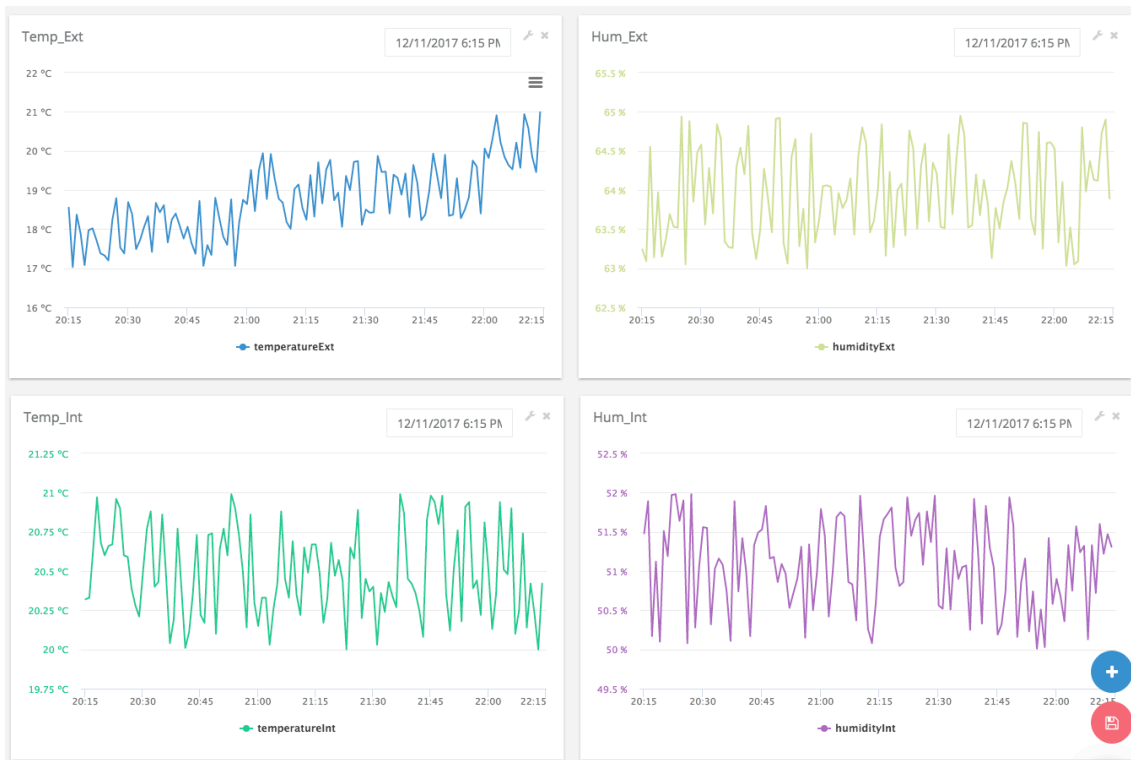


Figura 66. Dashboards distintos sensores.

En la sección Cloud Code se han definido un job y un Trigger como actuadores para la simulación del procesado en el Cloud.

Triggers

[Add Trigger](#)

Search...

Name	Created At	Product	Options
LowTempAlarmSms	2017-12-12T17:29:12.613Z	Raspi	EDIT DELETE

Jobs

[Add Job](#)

Search...

Name	Created At	Product	Options
TempExtData	2017-12-12T17:05:31.025Z	Raspi	EDIT DELETE

Figura 67. Job y Trigger.

El Job realiza la detección de un valor por debajo del umbral definido cada hora.

Job

Name: TempExtData

Frequency: Hourly

Code:

```
1
2 function job(params, callback){
3   analytics.events.getValuesByName('temperatureExt',function(error, data){
4     analytics.kpis.create('lowTemp',{ low: data });
5     callback();
6   });
7 }
8
9
```

SAVE Save And Continue Editing CANCEL

Figura 68. Job.

Mientras que el Trigger realizará el uso de la plataforma twilio[41] para enviarnos un sms indicando que el umbral de temperatura está por debajo del que hemos definido, este Trigger podría ser también un e-mail o un script realizando una petición WRITE de vuelta a nuestra Raspberry pi para ejecutar un script sobre un actuador, una placa Arduino o un puerto GPIO de la propia Raspberry pi.

Trigger

Name: LowTempAlarmSms

Code:

```
1 function trigger(params, callback){
2   if(params.action !== 'write') return callback()
3
4   var values = params.values
5
6   for(var i=0; i<values.length; ++i){
7     if(values[i].key === 'temperatureExt' && values[i].value < 19){
8       console.log('Low Temp Alarm')
9       var telf = '+34666111222'
10
11       var message = 'LOW TEMP ALARM!: ' + values[i].value
12
13       var twilio = new Twilio(
14         'AC4b3e35b39d9ac0d234073adcc8b23a8c',
15         '5b2a33ddb10e8a77bdfdd362083e22d4'
16       )
17
18       twilio.sendMessage({
19         to: telf,
20         from: '+34955160164',
21         body: message
22       },
23       callback
24     )
25   }
26 }
27 callback()*/
28 }
29
```

Figura 69. Trigger.

El resultado del Trigger es el siguiente:

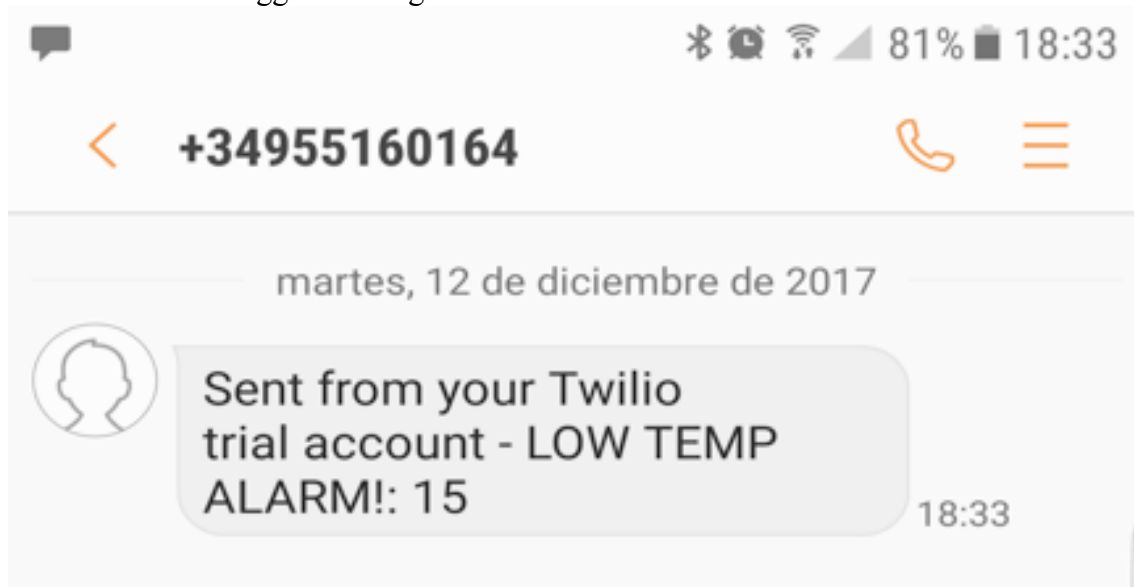


Figura 70. Resultado Trigger con Twilio.

4. Resultados comparativa Fog – Cloud Computing.

En esta parte se realiza la comparativa en las dos soluciones utilizadas en la simulación de computación.

Aunque la generación y envío de datos ya procesados se envían desde la Raspberry pi utilizando los mismos scripts hay que tener en cuenta los siguientes factores:

- Latencia conexión WAN: el envío de datos a la plataforma Cloud depende de entre distintos factores de nuestra línea arrendada a un ISP, el tiempo de respuesta medio por icmp es de aproximadamente unos 30ms, en la solución de fog computing donde solo utilizamos la plataforma para la representación de datos este tiempo es descartable, pero para cloud computing donde las tomas de decisiones se realizan en la nube este tiempo es crítico dependiendo de los actuadores que tengamos configurados.

```

[MBP-de-Ruben:~ Rubo$ nslookup thethings.io
Server:          10.101.193.117
Address:         10.101.193.117#53

Non-authoritative answer:
Name:   thethings.io
Address: 104.199.94.203

[MBP-de-Ruben:~ Rubo$ ping 104.199.94.203
PING 104.199.94.203 (104.199.94.203): 56 data bytes
64 bytes from 104.199.94.203: icmp_seq=0 ttl=58 time=29.847 ms
64 bytes from 104.199.94.203: icmp_seq=1 ttl=58 time=30.216 ms
64 bytes from 104.199.94.203: icmp_seq=2 ttl=58 time=29.565 ms
64 bytes from 104.199.94.203: icmp_seq=3 ttl=58 time=29.418 ms

```

Figura 71. Latencia a Thethings.io

- Seguridad: cuando el procesado y tratamiento de los datos se realizan mediante fog ninguna información sale de nuestra red local, todo está protegido mediante user/pass así como el acceso a la propia LAN, los datos procesados se envían en blanco por la WAN, en la solución Cloud todo es enviado a Thethings.io en blanco, no existe una solución de seguridad implantada para dotar de protección a los datos, aunque si existen plataformas como AWS donde se pueden definir certificados para las conexiones con o desde la plataforma a nuestro sistema.
- Disponibilidad Cloud: la plataforma Cloud puede estar indisponible por distintas causas, saturación del servidor, caída del servicio, etc. o por la pérdida de nuestra propia línea WAN, en la solución Fog siempre estará disponible el sistema a menos que haya una caída de nuestra LAN Ipv6 o algún tipo de problema en la Raspberry pi.
- Consumo de ancho de banda: cada paquete enviado tiene un peso según wireshark de aproximadamente unos 500 bytes, esto es aproximadamente unos 2Kb cada 60 segundos de información enviada al cloud 2,88Mb al día, para nuestro entorno es un consumo muy bajo, pero en redes que dispongan de un número elevado de dispositivos y en el que el payload sea mayor puede ser un problema dependiendo del ancho de banda contratado.
- Coste: la solución fog es la más económica, el ancho de banda hasta el procesado es el disponible en nuestra propia red LAN y todo el sistema se ha creado con hardware y software open-source lo que abarata la solución. Por el contrario al utilizar Cloud Computing necesitamos contar con una conexión a Internet arrendada a un ISP y una plataforma Cloud, para una aplicación IoT básica nos bastaría con Thethings.io pero si queremos añadir funcionalidades de bases de datos, actuadores, distintas clases de monitorización, etc. necesitamos una plataforma más potente como AWS o Google Cloud, las cuales tienen un coste asociado dependiendo del número de dispositivos, servicios a utilizar, número de llamadas desde nuestro dispositivo, etc.

5. Conclusiones

Durante el desarrollo de este trabajo fin de master se ha realizado una explicación de la situación del entorno IoT, desde su arquitectura, modelos, protocolos y campos de aplicación actual.

También se muestra dos tipos distintos de computación, fog computing y cloud computing así como distintos sistemas operativos existentes para IoT de los cuales se ha escogido Open-WSN para realizar la simulación de los sensores en nuestra red PAN.

Se han adquirido los conocimientos necesarios para poder desplegar nuestra propia red IPv6 y formar la conexión necesaria desde los motes al DAGroot para poder comunicar dichos sensores con nuestra plataforma donde está el sistema operativo instalado.

Por otra parte también se consigue conocer la estructura del sistema operativo para poder crear y enviar al hardware Openmote nuevas aplicaciones basadas en C para realizar distintas tareas con los sensores que tiene incorporados.

Durante la búsqueda de información de plataformas en la nube se encuentran diversos tipos, desde las gratuitas dependiendo del número de llamadas realizadas y con funciones limitadas, como realizar gráficas y scripts sencillos como actuadores a plataformas de pago como AWS o google cloud donde se encuentra todo tipo de servicios, máquinas virtuales, bases de datos no relacionales, etc.

Debido a problemas con la lectura de la información obtenida por los mote debido a problemas con la librería de CoAP Python se tiene que realizar una modificación de la planificación original siendo la información que se sube a la nube generada aleatoriamente mediante scripts en Python, estos scripts simulan la lectura de información obtenida de los sensores de temperatura y humedad de dos mote.

Una vez generados los datos aleatorios en la Raspberry Pi se realiza dos tipos de procesado, uno tipo fog donde la información se almacena en una base de datos tipo sql, se realizan las tomas de decisiones necesarias basándose en los valores obtenidos y se hace una subida de datos a la plataforma Thethings.io mediante una API en Python donde hacemos una representación gráfica de los resultados. Por otra parte la computación cloud, los datos una vez generados son subidos automáticamente a Thethings.io una vez ahí mediante un Job y un Trigger se realiza la toma de decisiones, en este caso se implementó utilizando twilio el envío de un sms indicando sí se había alcanzado una restricción en temperatura definida en el Trigger.

Ambas soluciones presentan un buen resultado para una solución doméstica donde se haga un control de temperatura, humedad, luminosidad, etc. y en base a los resultados se realicen una serie de acciones mediante actuadores, en cambio para una solución industrial donde el tiempo de actuación sea muy importante como por ejemplo un control del entorno para la seguridad de los trabajadores no puede existir un retardo entre la obtención de información y la actuación, por eso es más indicado una computación tipo fog para esta clase de entorno.

En líneas futuras para este proyecto podemos recoger las siguientes mejoras:

- Lectura real de los mote: realizar una depuración del problema encontrado con la librería CoAP y poder obtener una medida real, no simulada.
- Sistema de monitorización: instalación en la Raspberry Pi de una herramienta de monitorización como Zabbix o Cacti y que informe al usuario mediante e-mail de que se ha perdido la conectividad con los OpenMote.
- Uso de otras plataformas cloud: utilización de AWS o Google y almacenar sus datos utilizando nosql, análisis de datos, cálculo de máximos y mínimos, etc.
- Aplicación de filtros: si tenemos actuadores basándonos en las mediciones estos pueden estar continuamente conmutando de un estado on/off debido a errores en las medidas o a ligeras variaciones en el límite de las mismas, por eso sería buena idea implementar un filtro PID (Proporcional, integral y derivativo) bajo software para realizar un ajuste en las mediciones.

6. Glosario

ACK	Acknowledgement
API	Application Programing Interface
ARM	Avanced RISC Machine
ASK	Amplitude-shift keying
BPSK	Binary Phase Shift Keying
CFP	Contention Free Period
CoAP	Constrained Application Protocol
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DAO	Data Access Objects
DDS	Data-Distribution Service
DSSS	direct sequence spread spectrum
DTLS	Datagram Transport Layer Security
FFD	Full Function Device
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IPv6	Internet Protocol version 6
LLAP	Lightweight Local Automation Protocol
LLC	Logical Link Control
LWM2M	Light Weight Machine to Machine
M2M	Machine to Machine
MAC	Medium access control
MQTT	Message Queuing Telemetry Transport
NAT	Network address translation
OSI	Open System Interconnection
OVS	Open Visualizer
PaaS	Platform as a Service
PAN	Personal Area Network
PID	Proportional Integral Derivative
QPSK	Quadrature phase-shift keying
RAM	Random access memory
RFD	Reduced function device
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
SaaS	Software as a Service
Sms	Short Message Service
SSI	Simple Sensor Interface

TSMMP	Time Synchronized Mesh Protocol
UDP	User Datagram Protocol
uIP	Micro IP
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
XMPP	Extensible Messaging and Presence Protocol

7. Bibliografia

[1] Christopher Tozzi IoT Past and Present: The History of IoT, and Where It's Headed Today 2016 URL <http://talkincloud.com/cloud-computing/iot-past-and-present-history-iot-and-where-its-headed-today>

[2] Keith D. Foote A Brief History of the Internet of Things 2016 URL <http://www.dataversity.net/brief-history-internet-things/>

[3] Anna Gerber Simplify the development of your IoT solutions with IoT architectures 2017 URL <https://www.ibm.com/developerworks/library/iot-lp201-iot-architectures/index.html>

[4] Nomi Internet-of-Everything and Smart-cities Archive 2016 URL <http://www.telxperts.com/internet-of-everything/internet-everything-smart-cities-archive-2/>

[5] Daniel Karzel Hannelore Marginean Tuan-Si Tran A Reference Architecture for the Internet of Things 2016 URL <https://www.infoq.com/articles/internet-of-things-reference-architecture>

[6] <https://www.postscapes.com/internet-of-things-protocols/>

[7] kernelsphere THE FOUR INTERNET OF THINGS COMMUNICATION MODELS 2016 URL <https://www.kernelsphere.com/four-internet-things-communications-models/>

[8] <http://www.thewhir.com/web-hosting-news/the-four-internet-of-things-connectivity-models-explained>

[9] Teena Madox Smart cities: 6 essential technologies 2016 URL <https://www.techrepublic.com/article/smart-cities-6-essential-technologies/>

[10] Ciara O'Brien Wearables 2016 URL <https://www.irishtimes.com/business/technology/wearables-samsung-chases-fitness-fans-with-gear-fit-2-1.2763512>

[11] Andrew Meola How IoT and Smart home Automation will change the way we live 2017 URL <http://www.businessinsider.com/internet-of-things-smart-home-automation-2016-8>

[12] Paul Desmond Big Data and IoT promise big changes to agriculture 2016 URL <https://enterpriseproject.com/article/2016/8/big-data-and-iot-promise-big-changes-agriculture>

- [13] NXP IEEE 802.15.4 Stack User Guide 2016 URL <https://www.nxp.com/docs/en/user-guide/JN-UG-3024.pdf>
- [14] Giuseppe Anastasi From IEEE 802.15.4 to IEEE 802.15.4e 2014 URL <http://www.iet.unipi.it/g.anastasi/talks/2014-Guangzhou.pdf>
- [15] rfc7252 Internet Engineering Task Force 2014 URL <https://tools.ietf.org/html/rfc7252>
- [16] rfwireless-world What is CoAP IoT protocol I CoAP Architecture,message header 2016 URL <http://www.rfwireless-world.com/IoT/CoAP-protocol.html>
- [17] Margaret Rouse fog computing 2016 URL <http://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging>
- [18] Alexander Spotnitz Moving the Cloud to the Edge 2017 URL <https://www.pubnub.com/blog/moving-the-cloud-to-the-edge-computing/>
- [19] OpenFog DEFINITION OF FOG COMPUTING 2016 URL <https://www.openfogconsortium.org/resources/#definition-of-fog-computing>
- [20] Thierry Coupaye Cisco Fog Computing = Cloud + IoT 2014 URL <https://recherche.orange.com/en/fog-computing-and-geo-distributed-cloud/>
- [21] Arif Mohamed A history of cloud computing 2009 URL <http://www.computerweekly.com/feature/A-history-of-cloud-computing>
- [22] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli Fog Computing and Its Role in the Internet of Things 2016 URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.362.9132&rep=rep1&type=pdf>
- [23] <http://www.contiki-os.org/> 2017
- [24] <https://riot-os.org/> 2017
- [25] <https://mynewt.apache.org/> 2017
- [26] <https://openwsn.atlassian.net/wiki/spaces/OW/overview> 2017
- [27] Giuseppe Ribezzo OpenSim 2016 URL <https://openwsn.atlassian.net/wiki/spaces/OW/pages/13434892/OpenSim>

- [28] Thomas Watteyne CoAP Python Library 2014 URL <https://openwsn.atlassian.net/wiki/spaces/OW/pages/32014351/CoAP+Python+Library>
- [29] <https://dweet.io/> 2017
- [30] <https://freeboard.io/> 2017
- [31] Amazon. What is aws iot? 2016. URL <https://aws.amazon.com/es/iot/>
- [32] <https://thethings.io/> 2017
- [33] Thomas Watteyne, Xavier Vilajosana, Pere Tuset Open{WSNIMote}: Open-Source Industrial IoT 2016 URL <https://ercim-news.ercim.eu/images/stories/EN101/vilajosana2.png>
- [34] <http://gntoolchains.com/raspberry/jessie/> 2017
- [35] <https://github.com/openwsn-berkeley> 2017
- [36] <https://addons.mozilla.org/es/firefox/addon/copper-270430/> 2017
- [37] <https://www.raspberrypi.com/shop/es/raspberry-pi-3.php> 2017
- [38] Thomas Watteyne, Xavier Vilajosana, Pere Tuset Open{WSNIMote}: Open-Source Industrial IoT 2016 URL <https://ercim-news.ercim.eu/en101/special/open-wsn-mote-open-source-industrial-iot>
- [39] Xavier Vilajosana, Pere Tuset, Thomas Watteyne, Kris Pister OpenMote: Open-Source Prototyping Platform for the Industrial IoT 2015 URL https://www.researchgate.net/figure/An-OpenMote-CC2538-and-an-OpenUSB_280068237
- [40] <https://es.wikipedia.org/wiki/MariaDB> 2017
- [41] <https://www.twilio.com/> 2017